TREBALL DE FI DE MÀSTER

Màster en Enginyeria Matemàtica

Títol: Analytical and computational tools for the study of phase and synchrony in neural oscillators

Autor: Oriol Castejón

Directors: Antoni Guillamon i Joaquim Puig

Departament: Matemàtica Aplicada I

Convocatòria: Octubre 2010

Facultat de Matemàtiques i Estadística

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Treball de Fi de Màster

Analytical and computational tools for the study of phase and synchrony in neural oscillators

Oriol Castejón

Advisors: Antoni Guillamon i Joaquim Puig

Departament de Matemàtica Aplicada I

Agraïments

Vull agrair als meus tutors, el Toni i el Quim, que m'hagin ajudat tant a realitzar aquest treball, estant sempre pendents de l'evolució d'aquest i disposats a resoldre'm qualsevol dubte quan ho he necessitat, així com l'interès que m'han sabut transmetre per els temes que hem tractat.

També vull agrair a l'Adrià el seu suport incondicional, la tranquil·litat que m'ha sabut transmetre i que m'hagi escoltat sempre que ho he necessitat. D'altra banda, vull agrair als meus pares i a la meva germana la paciència que han tingut amb mi i l'interès que han mostrat sempre per saber de què anava aquest treball. Finalment vull agrair al Marcel els dubtes que m'ha solucionat durant tot aquest temps, i sobretot a tots els meus amics de la FME i del Costa, sense els quals res d'això hagués tingut sentit.

Abstract

This work aims both to give a broad vision of the applications of dynamical systems in mathematical neuroscience and to extend the tools and methods that are used up to now. In the first chapter, we will set the theoretical framework in which we will work, introducing the fundamental concepts of asymptothic phase, isochrons and phase resetting curves (PRCs), which can be defined in models of neural oscillators. With the aid of the theory of averaging we shall explain how the phase interaction functions of coupled oscillators can be derived from PRCs, specifying also how these functions can be used in synchronization problems.

Chapter 2 is devoted to the computational aspects of the previous chapter, using a powerful and versatile mathematical software called Sage. First of all we will explain the implementation of some standard routines in dynamical systems, namely the Poincaré map and its derivative, and how they can be used to find periodic orbits. Afterwards, we shall describe the implementation of the so-called Adjoint method, which is crucial for the computation of PRCs. Finally, we will present two contributions that we have made to the development of Sage: the adaptation of a routine to solve systems of ODEs numerically, combining the advantages of symbolic manipulation and the efficiency of numerical algorithms, and the creation of a Sage package from a simulator of brain dynamics called Brian. We will give examples of all the routines we created as well as the Brian package, to illustrate and clarify their use.

Finally, in the last chapter we shall introduce a new tool that we called second order PRCs, which are, in a way, an extension of the classical PRCs. First of all, we will focus on oscillators with pulsed coupling, deriving the second order PRCs in this case and presenting a method to compute them numerically. Finally, we will treat weakly perturbed and weakly connected networks of oscillators, which will lead us to introduce a new framework involving Poincaré-Lindsted series. In the case of pulsed coupling, we will give numerical examples of the computation of second order PRCs, and in the other case we will present analytical examples in which the role of this higher order PRC happens to be very relevant.

Key words: Phase resetting curves, synchronization, isochrons, Sage, mathematical neuroscience

MSC2000: 34C15, 92B25, 37M99, 34C25, 34C29, 65P40

Contents

Introduction

The purpose of this memoir is to expose some applications of dynamical systems in the field of mathematical biology, providing the basic ideas, presenting and developing computational methods, and finally contributing to extend the mathematical theory in some directions.

The main topic of this work are the so-called phase resetting curves (PRCs). The study of the phase of oscillators is becoming a popular subject, so in the first place we found interesting to study them in more depth. However, apart from being a very interesting mathematical object, it has many applications in time control problems in neuroscience. Moreover, they can be measured experimentally and deduced from mathematical models, which makes them interesting for both experimental and theoretical neuroscientists. Therefore, PRCs have been a perfect introduction to the fields of dynamical systems and neuroscience.

This memoir has the following structure: first of all we present the mathematical background of the theory of PRCs and their application to biological models, and more concretely to models related to neuroscience. Once the fundamental ideas and tools are defined, in Chapter 2 we will proceed to describe their numerical implementation using the mathematical software Sage, and give some examples. Besides, in this chapter we will also present a simulator of brain dynamics, called Brian, and explain how we adapted it to Sage. Finally, in the last chapter, we will present what we call second order PRCs, which are an extension of the ideas explained in the first chapter.

Chapter 1
Oscillators, phase resetting curves and synchronization

We shall now introduce the theoretical background in which PRCs are defined, proceeding as follows: first, we will introduce the concepts of phase, asymptotic phase and isochrons. Then we will explain the problem of the phase advance and PRCs, focusing in the next section on weakly connected networks. We will give a method to compute PRCs, and finally relate them to synchronization problems. We will follow mainly Chapter 10 of [**17**] and in some points [**8**], putting special emphasis on the mathematical justifications, which sometimes cannot be found in this literature. For an exhaustive survey of weakly connected networks, we refer the reader to [**16**].

1. Phase of an oscillator and isochrons

Consider a planar dynamical system of the form:

$$\dot{x} = f(x), \qquad x \in \mathcal{U} \subset \mathbb{R}^2, \tag{1}$$

where $f : \mathbb{R}^2 \to \mathbb{R}^2$ is a $\mathcal{C}^r$ field, $r \geq 1$. Suppose that (1) has a limit cycle of period T, say $\gamma = \{\gamma(t) \,|\, t \in [0, T)\}$. This limit cycle is parameterized by the variable $\vartheta = t$, $\vartheta \in [0, T)$, and its dynamics can be trivially described by the one-dimensional system:

$$\dot{\vartheta} = 1, \quad \vartheta \in [0, T). \tag{2}$$

We call ϑ the *phase* of γ.

Remark 1. It is also common to take the phase as $\vartheta = t/T$, $\vartheta \in [0, 1)$.

The phase can be extended in a neighborhood of the limit cycle using the following notions:

Definition 1. Let $\mathcal{U}$ be an open set containing the limit cycle γ. We say that a point $y \in \mathcal{U}$ is in *asymptotic phase* with a point $x \in \gamma$, and write $\vartheta(y) = \vartheta(x)$, if:

$$\lim_{t \to +\infty} |\varphi(t, y) - \varphi(t, x)| = 0,$$

or:

$$\lim_{t\to-\infty} |\varphi(t,y) - \varphi(t,x)| = 0,$$

where $\varphi(\cdot,\cdot)$ is the flow of (1).

DEFINITION 2. The set of points which are in phase with a point $x \in \gamma$, i.e. the set $\{y \in \mathcal{U} \mid \vartheta(y) = \vartheta(x)\}$, is called the *isochron* of x (sometimes it is also called the *stable manifold* of x). In other words, isochrons are the level curves of the function ϑ.

REMARK 2. One might wonder if isochrons always exist. The fact is that, while for hyperbolic limit cycles they do (see [**12**] for a proof), for non-hyperbolic cycles it is not so (see [**4**], [**12**] and [**7**] for examples). For instance, in [**4**] the authors show that for generic non-hyperbolic limit cycles isochrons exist in some neighborhood if and only if the derivative of the time of first return to some cross section Σ, $\tau(p)$, is zero; and in [**7**] the author solves completely the problem of the existence of asymptotic phase for analytic vector fields in $\mathbb{R}^2$.

Isochrons are invariant under the time T map, $\varphi(T,\cdot)$. Indeed, suppose that $y \in \mathcal{U}$ is in asympotic phase with $x \in \gamma$:

$$\lim_{t\to+\infty} |\varphi(t,y) - \varphi(t,x)| = 0$$

(the case $t \to -\infty$ is analogous). Then, because of the continuity of $\varphi(T,\cdot)$:

$$\lim_{t\to+\infty} |\varphi(T,\varphi(t,y)) - \varphi(T,\varphi(t,x))| = 0. \tag{3}$$

Using the properties of the flow we have that:

$$\varphi(T,\varphi(t,y)) = \varphi(t+T,y) = \varphi(t,\varphi(T,y)).$$

Besides, as x is on the limit cycle we have that:

$$\varphi(T,\varphi(t,x)) = \varphi(t,x).$$

Then, from (3) we conclude that:

$$\lim_{t\to+\infty} |\varphi(t,\varphi(T,y)) - \varphi(t,x)| = \lim_{t\to+\infty} |\varphi(T,\varphi(t,y)) - \varphi(T,\varphi(t,x))| = 0,$$

i.e. $\varphi(T,y)$ is on the isochron of x.

Similarly, it can be shown that isochrons are mapped to isochrons by the flow φ: suppose that x and y lie on the same isochron. Then for all s we have:

$$0 = \lim_{t\to+\infty} |\varphi(s,\varphi(t,y)) - \varphi(s,\varphi(t,x))| = \lim_{t\to+\infty} |\varphi(t,\varphi(s,y)) - \varphi(t,\varphi(s,x))|,$$

that is, $\varphi(s,y)$ and $\varphi(s,x)$ are also in asymptotic phase.

REMARK 3. As a consequence, we have that equation (2) is satisfied for all $\vartheta(y)$, for y in some open set $\mathcal{V} \supset \gamma$: suppose that $x \in \gamma \subset \mathcal{V}$, $y \in \mathcal{V}$ and $\vartheta(x) = \vartheta(y)$, then we know that $\vartheta(\varphi(t,x)) = \vartheta(\varphi(t,y))$. In conclusion, we have that:

$$\dot{\vartheta}(\varphi(t,y)) = \dot{\vartheta}(\varphi(t,x)) = 1.$$

REMARK 4. In [**25**], it is shown that isochrons are defined in a neighborhood of the limit cycle if and only if there is a vector field Y transversal to f and a function $\mu : \mathbb{R}^2 \to \mathbb{R}$ such that:

$$[Y, f] = \mu Y,$$

where $[\cdot, \cdot]$ is the Lie bracket. Moreover, the isochrons of f are the orbits of the system:

$$y' = Y(y).$$

In general this result does not help to compute the isochrons, because it is difficult to find such Y and μ, but in the examples of Chapter 3 we will use it.

2. Phase resetting curves

Systems like (1) can be used to describe a periodically spiking neuron, and then ϑ is usually defined so that the spikes occur at $\vartheta = 0$. We will now focus on models that describe the behavior of this neuron when a brief pulse of current is applied at a given time t_s in some direction $v = (v_1, v_2) \in \mathbb{R}^2$, increasing the first state variable by v_1 and the second one by v_2. In fact, this is just like a change of "initial conditions" at time $t = t_s$. This can be written mathematically by:

$$\dot{x} = f(x) + v\delta(t - t_s),$$

where $\delta(t)$ is the Dirac delta function.

REMARK 5. Let us see that indeed the difference between the state variables right before t_s and at t_s is v. This difference is given by:

$$x(t_s) - \lim_{h \to 0^+} x(t_s - h).$$

Writing $x(t)$ in its integral form, we have that:

$$\begin{aligned} x(t_s) - \lim_{h \to 0^+} x(t_s - h) &= \int_0^{t_s} \dot{x}(t)dt + x(0) - \lim_{h \to 0^+} \left(\int_0^{t_s - h} \dot{x}(t)dt + x(0) \right) = \\ &= \lim_{h \to 0^+} \left(\int_{t_s - h}^{t_s} \dot{x}(t)dt \right) = \lim_{h \to 0^+} \left(\int_{t_s - h}^{t_s} \left[f(x(t)) + v\delta(t - t_s) \right] dt \right) = \\ &= v + \lim_{h \to 0^+} \left(\int_{t_s - h}^{t_s} f(x(t))dt \right) = v. \end{aligned}$$

As we already mentioned in the introduction, in neuroscience it is important to know how the neuron will behave after such a perturbation, $v\delta(t - t_s)$, at a point of the limit cycle $x_{t_s} = \gamma(t_s)$. More precisely, it is relevant to know when the next spike will occur, and compare it with the case when the neuron is not stimulated. In terms of the phase, one has to compute the phase of the unperturbed system at time t_s, say $\vartheta(x_{t_s})$, and see if it is greater or lower than the phase of the perturbed system, $\vartheta(x_{t_s} + v)$.

The so-called *phase resetting curve* or *phase response curve* (PRC) is the function that, to each phase $\vartheta(x_{t_s})$, assigns the phase advance obtained by applying a certain impulse at time $t = t_s$. That is:

$$PRC(\vartheta(x_{t_s})) = \vartheta(x_{t_s} + v) - \vartheta(x_{t_s}).$$

Experimentally, one can obtain the PRC by stimulating the neuron at different phases and measuring the new phase once the neuron has relaxed back to the limit cycle. PRCs are a powerful tool in time-control problems, which are common in mathematical biology. For example, they can be used to control circadian rhythms, because they provide a way to predict the effects of some sleep-altering drugs depending on the time when they are given to the patient (see for instance [**2**], [**5**] and [**22**]).

Using mathematical models, PRCs can be computed noting that:

$$\vartheta(x_{t_s} + v) - \vartheta(x_{t_s}) = \nabla\vartheta(x_{t_s}) \cdot v + O(|v|^2),$$

so that for small stimuli the PRC can be approximated by $\nabla\vartheta(x_{t_s}) \cdot v$; that is, the scalar product between the orthogonal vector to the isochron of x_{t_s} and the stimulus v. For this reason, the function $\nabla\vartheta$ is called *infinitesimal phase resetting curve* (iPRC), although it is usually referred to simply as PRC, and we shall do it in the following.

Remark 6. There is recent work that extends the theory of PRCs in a neighborhood of the limit cycle. Its interest lies on the fact that for high-frequency stimuli or other typical phenomena of neuronal activity, such as *bursting* and noise, the dynamics can take place "far" from the asymptotic state. In this case, instead of having Phase Resetting *Curves*, one has Phase Resetting *Surfaces* (see [**13**]).

3. The Adjoint Method

The most common method to compute PRCs is the so-called *Adjoint Method*, which we will describe in this section. We will follow the standard proof found in [**1**]. However, a more general proof can be found in [**13**].

Theorem 1. *If $\gamma(t)$ is a T-periodic solution of the unperturbed system* (1)*, then $\nabla\vartheta(\gamma(t))$ is the T-periodic solution of the adjoint equation:*

$$\dot{Q} = -Df^T(\gamma(t))Q, \tag{4}$$

satisfying the normalization condition:

$$Q(t) \cdot f(\gamma(t)) = 1. \tag{5}$$

Proof. Consider the limit cycle $\gamma(t)$ of the unperturbed system (1), and let $x(t)$ be the solution of (1) with initial condition $\gamma(0)+\Delta x$. If we define $\Delta x(t) = x(t)-\gamma(t)$, we have that:

$$\frac{d\Delta x(t)}{dt} = Df(\gamma(t))\Delta x(t) + O(||\Delta x||^2). \tag{6}$$

Defining the phase shift as $\Delta\vartheta = \vartheta(x(t)) - \vartheta(\gamma(t))$, we can write:

$$\Delta\vartheta = \nabla\vartheta(\gamma(t)) \cdot \Delta x(t) + O(||\Delta x||^2), \tag{7}$$

where $\cdot$ stands for the ordinary dot product. Recalling that $\Delta\vartheta$ is independent of time (by Remark 3) and taking the time derivative of equation (7), to lowest order in $||\Delta x||$ we get:

$$\frac{d\nabla\vartheta(\gamma(t))}{dt} \cdot \Delta x(t) = -\nabla\vartheta(\gamma(t)) \cdot \frac{d\Delta x(t)}{dt}.$$

Using (6) we have:

$$\frac{d\nabla\vartheta(\gamma(t))}{dt}\cdot\Delta x(t) = -\nabla\vartheta(\gamma(t))\cdot Df(\gamma(t))\Delta x(t) = -Df^T(\gamma(t))\nabla\vartheta(\gamma(t))\cdot\Delta x(t).$$

Finally, as this last equality must hold for arbitrary perturbations, we have:

$$\frac{d\nabla\vartheta(\gamma(t))}{dt} = -Df^T(\gamma(t))\nabla\vartheta(\gamma(t)).$$

Finally, from (2) we already know that:

$$\nabla\vartheta(\gamma(t))\cdot f(\gamma(t)) = 1.$$

□

REMARK 7. In practice, what we will do is to look for T-periodic solutions of (4) whose initial conditions satisfy the normalization condition (5) (see Chapter 2 to see how it can be done). Imposing the T-periodicity is not sufficient, since (4) is a linear system and hence, if Q is a T-periodic solution, $\tilde{Q} = \lambda Q$ will be also a T-periodic solution for all $\lambda \in \mathbb{R}$, $\lambda \neq 0$. This leaves one degree of freedom to choose the initial condition, and therefore we need (5) to find the *right one*.

4. Weak coupling and PRCs

Consider now a more general perturbation:

$$\dot{x} = f(x) + \varepsilon g(t,x). \tag{8}$$

where g is a $\mathcal{C}^r$ field, $r \geq 1$, and suppose that for ε sufficiently small it still has a limit cycle. For example, this could represent a model of a neuron weakly coupled with other neurons in a network. What we would like to do now is to obtain a phase model similar to (2) for the perturbed system (8). We will follow a procedure based on that introduced by Kuramoto in [**18**].

First consider the unperturbed system (1), with solutions $x = x(t)$. Differentiating $\vartheta(x)$ with respect to t we have that:

$$\frac{d\vartheta(x)}{dt} = \nabla\vartheta(x)\cdot\frac{dx}{dt} = \nabla\vartheta(x)\cdot f(x).$$

However, by (2) and Remark 3 we know that in a neighborhood $\mathcal{U}$ of the limit cycle:

$$\frac{d\vartheta(x)}{dt} = 1.$$

Therefore, we conclude that:

$$\nabla\vartheta(x)\cdot f(x) = 1. \tag{9}$$

Note that this equality holds for every point in $x \in \mathcal{U}$ (without having to suppose that $x = x(t)$ is a solution of (1)).

Now, let $p_0 \in \gamma$ and Σ its isochron. For ε small enough, the limit cycle of the perturbed system (8) intersects Σ at some point $p(\varepsilon)$. Let $x_\varepsilon = \varphi(t, p(\varepsilon), \varepsilon)$ be a

solution of (8) with initial condition $p(\varepsilon)$. Differentiating again $\vartheta(x_\varepsilon)$ with respect to t we have:

$$\begin{aligned}\frac{d\vartheta(x_\varepsilon)}{dt} &= \nabla\vartheta(x_\varepsilon)\cdot\frac{dx_\varepsilon}{dt} = \nabla\vartheta(x_\varepsilon)\cdot(f(x_\varepsilon)+\varepsilon g(t,x_\varepsilon)) = \\ &= \nabla\vartheta(x_\varepsilon)\cdot f(x_\varepsilon)+\varepsilon\nabla\vartheta(x_\varepsilon)\cdot g(t,x_\varepsilon).\end{aligned}$$

Using (9) we obtain the phase model for (8):

$$\frac{d\vartheta(x_\varepsilon)}{dt} = 1+\varepsilon\nabla\vartheta(x_\varepsilon)\cdot g(t,x_\varepsilon). \tag{10}$$

However, we would like to get an approximation of (10) in terms of ε. Writing x_ε as:

$$x_\varepsilon(t) = x_0(t)+O(\varepsilon),$$

the Taylor series for ϑ and g yield:

$$\vartheta(x_\varepsilon) = \vartheta(x_0)+O(\varepsilon)$$

$$g(t,x_\varepsilon) = g(t,x_0)+O(\varepsilon)$$

Replacing it in (10) we can rewrite this equation as:

$$\frac{d\vartheta(x_\varepsilon)}{dt} = 1+\varepsilon\nabla\vartheta(x_0)\cdot g(t,x_0)+O(\varepsilon^2). \tag{11}$$

Note that $x_0(t)$ is the solution of the unperturbed system (1) with initial condition on the isochron of $p(\varepsilon)$. That is, with the convenient parameterization, we can write:

$$x_0(t) = \gamma(t).$$

Then (11) reads:

$$\dot\vartheta = 1+\varepsilon\nabla\vartheta(\gamma(t))\cdot g(t,\gamma(t))+O(\varepsilon^2), \tag{12}$$

obtaining again the PRC.

The relevance of (12) is that it allows us to approximate the phase of the solutions. However, as we will see in the next section, it can also be useful to determine phase-locking states of networks of coupled neurons.

5. Averaging the phase equations

Now we will consider systems of n neurons, $x_1,\dots,x_n$, with couplings given by some functions $g_{ij}(x_i,x_j)$:

$$\dot x_i = f(x_i)+\varepsilon\sum_{j=1}^{n} g_{ij}(x_i,x_j), \qquad i=1,\dots,n. \tag{13}$$

We will also assume that in the unperturbed system (that is, when $\varepsilon=0$) each oscillator has a periodic orbit of the same period T. Then, following a procedure similar to that of Section 4 we obtain the phase model:

$$\dot\vartheta_i = 1+\varepsilon Q_i(\vartheta_i)\cdot\sum_{j=1}^{n} g_{ij}(x_i(\vartheta_i),x_j(\vartheta_j))+O(\varepsilon^2), \qquad i=1,\dots,n, \tag{14}$$

where $x_i(\vartheta_i)$ is the point on the i-th limit cycle having phase ϑ_i and $Q_i(\vartheta_i) = \nabla\vartheta_i(x_i(\vartheta_i))$. We would like to apply the theory of averaging, but first of all we need to do a change of variables. Suppose that $\vartheta_i(t)$ is of the form:

$$\vartheta_i(t) = t + \varphi_i(t), \qquad i = 1, \ldots, n.$$

Then, substituting it in (14) we have that:

$$\dot{\varphi}_i = \varepsilon Q_i(t+\varphi_i) \cdot \sum_{j=1}^{n} g_{ij}(x_i(t+\varphi_i), x_j(t+\varphi_j)) + O(\varepsilon^2) \qquad i = 1, \ldots, n. \tag{15}$$

Note that (15) is a periodic system, and therefore it is possible to apply the theory of periodic averaging. For the sake of clarity we present here a general theorem of averaging as found in [**26**]. For its proof and more details, we refer the reader to the same book.

THEOREM 2. *Consider the initial value problems:*

$$\begin{cases} \dot{x} &= \varepsilon f(t,x), \\ x(0) &= x_0, \end{cases}$$

with $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$, *and:*

$$\begin{cases} \dot{y} &= \varepsilon f^0(x), \\ y(0) &= x_0, \end{cases}$$

x,y,$x_0 \in D \subset \mathbb{R}^n$, $t \in [0, \infty)$, $\varepsilon \in (0, \varepsilon_0]$. *Suppose:*

a) f *is* T*-periodic in* t*;*
b) f *is Lipschitz-continuous in* x *on* $D \subset \mathbb{R}^n$, $t \geq 0$, *continuous in* t *and* x *on* $\mathbb{R}^+ \times D$ *and with average* f^0, *that is:*

$$f^0(x) = \frac{1}{T} \int_0^T f(t,x)dt;$$

c) $y(t)$ *belongs to an interior subset of* D *on the time-scale* $\dfrac{1}{\varepsilon}$ *(that is, there exists a constant* L *such that* $y(t) \in \mathcal{U} \subset D$ *for* $0 \leq \varepsilon t \leq L$*);*

then:

$$x(t) - y(t) = O(\varepsilon),$$

as $\varepsilon \downarrow 0$ *on the time-scale* $\dfrac{1}{\varepsilon}$.

Applying this theorem, system (15) can be approximated by:

$$\dot{\varphi}_i = \varepsilon\omega_i + \varepsilon \sum_{j \neq i} H_{ij}(\varphi_j - \varphi_i), \qquad i = 1, \ldots, n, \tag{16}$$

where:

$$H_{ij}(\varphi_j - \varphi_i) = \int_0^T \nabla\vartheta_i(t) \cdot g_{ij}(x_i(t), x_j(t + \varphi_j - \varphi_i))dt,$$

and:

$$\omega_i = H_{ii}(\varphi_i - \varphi_i) = H_{ii}(0).$$

System (16) is useful because it is autonomous, and therefore more treatable. On the other hand, as we will see in the next subsection, in case of two coupled oscillators it can be used to tell the existence of states of phase-locking and their stability.

5.1. Synchronization of two oscillators. Consider (15) with $n = 2$, that is, the phase model of 2 coupled oscillators. Let us introduce a new independent variable, the *slow time*, $\tau = \varepsilon t$. Then, system (16) can be written as:

$$\begin{aligned} \varphi_1' &= \omega_1 + H_1(\varphi_2 - \varphi_1), \\ \varphi_2' &= \omega_2 + H_2(\varphi_1 - \varphi_2), \end{aligned} \tag{17}$$

where $' = d/d\tau$ is the derivative with respect to τ. Our aim is to derive a new system from (17) that describes the phase difference and its behavior through time.

Define the phase difference $\chi = \varphi_2 - \varphi_1$; then we have that:

$$\chi' = \omega + G(\chi), \tag{18}$$

where:

$$\omega = \omega_2 - \omega_1, \qquad \text{and} \qquad G(\chi) = H_2(-\chi) - H_1(\chi).$$

Equilibrium points χ^* of (18) are given by:

$$G(\chi^*) = -\omega,$$

and they correspond to phase locking states of the coupled system. In other words, if two oscillators have an initial phase difference equal to χ^*, it will remain so forever, and they are said to be *synchronized*. Moreover, the stability of this equilibrium points (which is given by the derivative of G) tells us if, for a phase difference close to χ^*, the system will tend to the synchronization state (in case that $G'(\chi^*) < 0$, that is, χ^* is stable) or not (in case that $G'(\chi^*) > 0$).

REMARK 8. If the two identical oscillators are coupled symmetrically, then $G(\chi) = H(-\chi) - H(\chi)$, and there always exist the equilibria $\chi^* = 0$ and $\chi^* = T/2$.

Now that we have presented the theoretical background of isochrons, PRCs and synchrony, we shall proceed to explain how to do the numerical computation of some of the tools we have seen. Later in Chapter 3, we will extend the concept of PRC, explain how this extension can be computed and illustrate in which cases it can represent an improvement with respect to the classical PRC.

Chapter 2
Developing computational tools with Sage

In this chapter we will present the numerical implementation of the ideas introduced in Chapter 1. Let us explain first of all what is Sage and why we have chosen it.

Sage (see [**28**]) is an open source mathematical software system based on Python, which is a powerful and efficient programming language, being a very suitable alternative to other software like Matlab, Maple or Mathematica. One of the main reasons that made it attractive for us was the possibility to contribute to it, that is, to create routines that would actually be present in future versions of the software. Another great feature of Sage, which cannot be found in other mathematical packages, is the possibility of combining the simplicity of symbolic manipulation and the precision and efficiency of numerical computations.

We had two main goals in mind. First of all, we wanted to create a set of routines related to dynamical systems and, more specifically, to the computation of PRCs and its applications to synchronization of oscillators. As PRCs are defined for limit cycles, we began by creating a function to find a periodic orbit of a given 2-dimensional dynamical system, and while developing it, it turned out that other functions were needed, such as the Poincaré map and the computation of its derivative. Finally, we implemented the computation of the adjoint (that is, the PRC), the interaction function H, and its derivative, which tell us whether a phase-locking state exists and its stability.

We must say that, on the way, we realized that it would be interesting to create some routines to combine symbolic functions and efficient integration of ODEs, and decided to share them with the whole Sage community, and we will also present it in this memoir.

The second goal we had in mind from the beginning was to integrate Brian, a Python-based simulator for neuronal dynamics (see [**10**] and [**11**] for a survey), in the context of Sage. We found this simulator interesting because it provides both an efficient and comfortable way to simulate the dynamics of large networks of neurons. Besides, the fact that it is written in Python made us think that the task of combining the two platforms would be relatively simple. Thus we produced

a package that can be downloaded from the Sage main web page and installed, contributing in a new way to the development of this mathematical software.

Throughout this chapter we will give examples of the use of the routines we have created as well as that of the Brian package.

1. Periodic orbits and the Poincaré map

The ideas introduced in this section can be found in many standard books of differential equations, for example [**3**], from a theoretical point of view, or [**14**], from a rather computational one. For some background on numerical analysis, see [**29**].

Consider a dynamical system given by an ODE:

$$\dot{x} = f(x), \qquad x \in \mathbb{R}^n,\ f \in \mathcal{C}^r,\ r \geq 1. \tag{19}$$

We want to find a periodic solution of (19), that is, a solution $x_p(t)$ such that there exists $T > 0$ verifying that $x_p(t+T) = x_p(t)$ for all t. Clearly, finding just one $x^* \in \gamma = \{x_p(t) | t \in [0, T)\}$ is sufficient, since the periodic solution will be $\varphi(t, x^*)$, that is, the solution of (19) with initial condition x^*. We will see that the problem of finding such a point can be reduced to that of finding a zero of a given function, for which we can use the well known Newton's method.

First of all, let us recall the concept of the Poincaré map.

1.1. The Poincaré map. Let Σ be a cross section of the vector field f in $x_0 \in \mathbb{R}^n$, that is, a hypersurface containing x_0 such that for all $x \in \Sigma$, $f(x) \notin T_x\Sigma$. Suppose that for some $T > 0$, we have $\varphi(T, x_0) \in \Sigma$ and $\varphi(t, x_0) \notin \Sigma$ for all $0 < t < T$.

Definition 3. If $\mathcal{U}$ is a neighborhood of x_0, we define the Poincaré map:

$$\begin{array}{rccc} \mathcal{P}: & \mathcal{U} \cap \Sigma & \longrightarrow & \Sigma \\ & x & \longmapsto & \mathcal{P}(x) = \varphi(\tau(x), x) \end{array}$$

where $\tau(x)$ is the time that takes x to return to Σ.

Remark 9. This map is indeed well defined. If Σ is characterized by the equation $h(x) = 0$, then we have:

$$Dh(x)f(x) \neq 0, \quad x \in \Sigma \tag{20}$$

because Σ is a cross section of f. Since $h(\varphi(T, x_0)) = 0$, $\tau(x)$ is obtained in a neighborhood of x_0 by applying the Implicit Function Theorem to the equation:

$$h(\varphi(\tau, x)) = 0,$$

whose derivative with respect to τ is precisely (20), and therefore non-zero in Σ.

Numerically, we implemented the Poincaré map as follows. Given an initial condition $x_0 \in \Sigma$, we integrate the ODE until the solution has crossed the section, so we are sufficiently close to it. Then we apply Newton's method to the function:

$$G(t) = h(\varphi(t, x_0)),$$

which yields the successive approximations:

$$t_{k+1} = t_k - \frac{G(t_k)}{G'(t_k)}.$$

Note that:

$$G'(t) = Dh(\varphi(t, x_0))\frac{d\varphi}{dt}(t, x_0) = Dh(\varphi(t, x_0))f(\varphi(t, x_0)).$$

1.2. Finding a periodic orbit. We now return to the task of finding a periodic orbit γ. It is clear that if $x^* \in \gamma \cap \Sigma$, then:

$$\mathcal{P}(x^*) = x^*,$$

or equivalently:

$$\mathcal{P}(x^*) - x^* = 0.$$

So, the only thing we must do in order to find a point of the periodic orbit is to take a cross section Σ that intersects this orbit, and look for a zero of the function:

$$\mathcal{F} = \mathcal{P} - Id.$$

This can be done by Newton's method again. Given a starting point x_0, the successive approximations are given by the following expression:

$$x_{k+1} = x_k - D\mathcal{F}^{-1}(x_k)\mathcal{F}(x_k).$$

In this case, we have that:

$$D\mathcal{F} = D\mathcal{P} - I_n$$

where I_n is the $n \times n$ Identity matrix. However, we have to work out how to compute the derivative of the Poincaré map, $D\mathcal{P}$, because obviously we do not know the map explicitly.

Recall that $\mathcal{P}(x) = \varphi(\tau(x), x)$. Then differentiation with respect to x gives:

$$D\mathcal{P}(x) = \frac{\partial\varphi}{\partial t}(\varphi(\tau(x), x))D\tau(x) + \frac{\partial\varphi}{\partial x}(\tau(x), x) = \underbrace{f(\varphi(\tau(x), x))}_{(1)}\underbrace{D\tau(x)}_{(3)} + \underbrace{\frac{\partial\varphi}{\partial x}(\tau(x), x)}_{(2)}.$$

These three terms can be determined as follows:

(1) $f(\varphi(\tau(x), x))$: it is trivially computed once we have implemented numerically the Poincaré map.

(2) $\dfrac{\partial\varphi}{\partial x}(\tau(x), x)$: this term can be found solving the first order variational equations, that is, finding a matrix solution of the initial value problem:

$$\begin{cases} \dot{Y} &= Df(\varphi(t, x_0))Y, \\ Y(0) &= I_n. \end{cases}$$

(3) $D\tau(x)$: one must do a little bit more work to compute this term. By definition of $\tau(x)$ and Σ, we have that:

$$h(\varphi(\tau(x), x)) = 0,$$

and differentiation with respect to x gives:

$$Dh(\varphi(\tau(x),x))\left[f(\varphi(\tau(x),x))D\tau(x)+\frac{\partial\varphi}{\partial x}(\tau(x),x)\right]=$$

$$=\left[Dh(\varphi(\tau(x),x))f(\varphi(\tau(x),x))\right]D\tau(x)+Dh(\varphi(\tau(x),x))\frac{\partial\varphi}{\partial x}(\tau(x),x)=0,$$

where one should have in mind that:

$$Dh=\left(\frac{\partial h}{\partial x_1}\cdots\frac{\partial h}{\partial x_n}\right),$$

$$D\tau=\left(\frac{\partial \tau}{\partial x_1}\cdots\frac{\partial \tau}{\partial x_n}\right),$$

and:

$$f=\begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}.$$

Then:

$$D\tau(x)=\frac{-1}{Dh(\varphi(\tau(x),x))f(\varphi(\tau(x),x))}Dh(\varphi(\tau(x),x))\frac{\partial\varphi}{\partial x}(\tau(x),x).$$

Note that the right terms of the last expression are all known.

In conclusion, we have that:

$$D\mathcal{P}(x)=\frac{-f(\varphi(\tau(x),x))}{Dh(\varphi(\tau(x),x))f(\varphi(\tau(x),x))}Dh(\varphi(\tau(x),x))\frac{\partial\varphi}{\partial x}(\tau(x),x)+\frac{\partial\varphi}{\partial x}(\tau(x),x).$$

REMARK 10. It is worth saying that in general, the computation of the Jacobian matrix of f (which is needed to solve the variational equations) has to be done by hand. However, using Sage symbolic functions, the Jacobian can be (exactly) computed with a simple command, and so we can create a routine to compute directly the derivative of the Poincaré map, without requiring the user to specify the Jacobian matrix of the field f nor using finite-difference routines.

2. Solving the adjoint system

Now we will focus on the numerical solution of the Adjoint method, i.e. the solution of the system:

$$\dot{Q}=-Df^T(x(t))Q, \tag{21}$$

where $x(t)$ is a periodic solution of period $T>0$.

The initial conditions of Q must be such that:

(a) $Q(t+T)=Q(t)$ for all t. That is, $Q(t)$ must be T-periodic.
(b) $Q(0)\cdot f(x(0))=1$, where $\cdot$ stands for the standard dot product.

First of all, note that if $Q(t)$ is any solution of (21) and $\lambda \in \mathbb{R}$, then $\lambda Q(t)$ is also a solution of (21). Therefore, if we find a periodic solution of (21), say $\tilde{Q}(t)$, and define:

$$\lambda = \frac{1}{\tilde{Q}(0) \cdot f(x(0))},$$

then $Q(t) = \lambda \tilde{Q}(t)$ will be a solution of the adjoint system satisfying conditions (a) and (b).

REMARK 11. It is easy to see that for any T-periodic solution $\tilde{Q}(t)$, $\tilde{Q}(0) \cdot f(x(0)) \neq 0$. Indeed, if $\tilde{Q}(0) \cdot f(x(0)) = 0$, it must be $\tilde{Q}(0) = 0$, and then $\tilde{Q}(t) = 0$ for all t which is a contradiction with the T-periodicity of $\tilde{Q}$ ($T > 0$), or $f(x(0)) = 0$ and then $x(0)$ would be a critical point of (19), and $x(t)$ would not be a T-periodic solution of this system.

Hence, we just have to focus on finding a periodic solution of the adjoint system. We could do it by following the method explained in the preceding section, but in this case we can follow a much more efficient procedure because of the linearity of (21).

Given a linear system of ODEs, we call a matrix $\Phi(t)$ fundamental matrix of the system if its columns are solutions of the system (i.e., $\Phi(t)$ is a matrix solution of the system) and they are linearly independent. If $\Phi(t)$ is such that $\Phi(0) = I_n$, then it is clear that $\Phi(t)x_0 = \varphi(t, x_0)$. On the other hand, if x_0 is a point of a T-periodic orbit, $\varphi(T, x_0) = x_0$, and thus:

$$\Phi(T)x_0 = x_0.$$

As a consequence of this last expression, we have that x_0 is a point of a T-periodic orbit if and only if it is an eigenvector of eigenvalue one of the matrix $\Phi(T)$. Therefore, in this case, to find x_0 we do not have to use Newton's method but instead solve the (matrix) initial value problem:

$$\begin{aligned} \dot{\Phi} &= -Df^T(x(t))\Phi, \\ \Phi(0) &= I_n. \end{aligned}$$

at time T, obtaining the matrix $\Phi(T)$, and computing numerically its eigenvector of eigenvalue one x_0. Once we have the initial condition, it just remains to solve (21) with any standard ODE solver.

3. The interaction function and synchronization

Once we have a routine to compute the adjoint of a given system, the interaction function is computed easily.

Recall that coupling n oscillators with natural frequencies ω_i leads to the phase interaction functions H_{ij}, which are:

$$H_{ij}(\varphi_i, \varphi_j) = \frac{1}{T} \int_0^T Q_i(t) \cdot g_{ij}(x_i(t), x_j(t + \varphi_j - \varphi_i))dt.$$

Hence, having the value of Q_i at some points $t_1, \ldots, t_n$ and computing the value of g_{ij} at these same points (for some fixed φ_i, φ_j), the integral can be divided into:

$$H_{ij}(\varphi_i, \varphi_j) = \frac{1}{T} \sum_{k=1}^{n-1} \int_{t_k}^{t_{k+1}} Q_i(t) \cdot g_{ij}(x_i(t), x_j(t + \varphi_j - \varphi_i))dt.$$

Each integral can be then computed by a standard quadrature, for example by Simpson's rule. In this case, the error that is made in the computation of each integral is:

$$e_k = \frac{(t_{k+1} - t_k)^5}{90} \left| f_{ij}^{(4)}(\xi, \varphi_i, \varphi_j) \right|,$$

where $\xi \in [t_k, t_{k+1}]$ and:

$$f_{ij}(t, \varphi_i, \varphi_j) = Q_i(t) \cdot g_{ij}(x_i(t), x_j(t + \varphi_j - \varphi_i)).$$

On the other hand, in the case of just two coupled oscillators we had that the phase difference χ could be described by:

$$\chi' = \omega + G(\chi),$$

where:

$$\omega = \omega_1 - \omega_2,$$

and:

$$G(\chi) = H_{21}(-\chi) - H_{12}(\chi).$$

As we have seen in Chapter 1, the function G may be used to determine phase locking or synchronization states and their stability. Having the routine to compute the function H_{ij}, it is trivial to compute the function G.

4. Examples

In this section we will present four examples of the use of the routines we explained before with systems which are, more or less directly, related to neuroscience.

4.1. The Andronov-Hopf oscillator. We begin by the Andronov-Hopf oscillator:

$$\left\{ \begin{array}{rcl} \dot{x} & = & x - y - x(x^2 + y^2), \\ \dot{y} & = & x + y - y(x^2 + y^2). \end{array} \right. \tag{22}$$

We define the equations in Sage by introducing:

```
x,y = var('x,y')    # Declare symbolic variables x and y
ah1 = x-y-x**3-2*x*y**2+y**2*x
ah2 = x+y+x**2*y-y**3-2*x**2*y
ah = [ah1,ah2]
```

We take some initial conditions `ic` and use the routine `find_po` (see Appendix A.3) to find the intersection of the periodic orbit with the Poincaré section $y =$`ic[1]` (the second component of `ic`) and its period.

```
ic = [2.0,0.0]
icah, periodah = find_po(ah,ic)
```

The result is (see also Figure 1):

$$\texttt{icah} = [\texttt{1.0}, \texttt{0.0}]\,, \qquad \texttt{periodah} = \texttt{6.28318530718}.$$

Now, we solve the adjoint system with the routine `adjoint`:

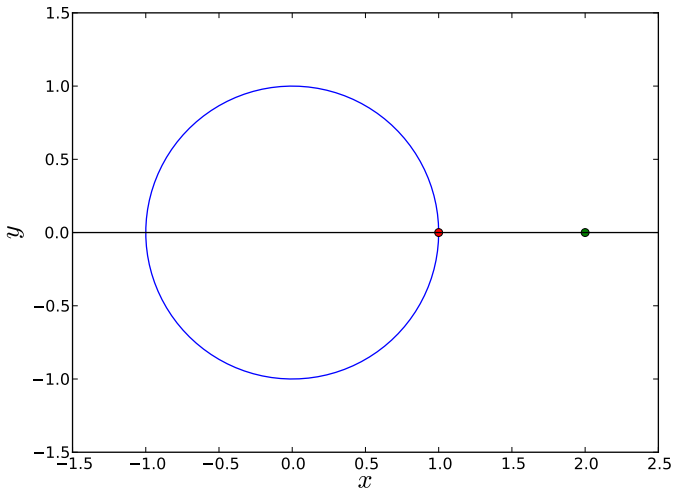


FIG. 1. Periodic orbit of the Andronov-Hopf oscillator (22). In green, `ic`. In red, `icah`.

```
aht,ahq = adjoint(ah,icah,periodah)
```

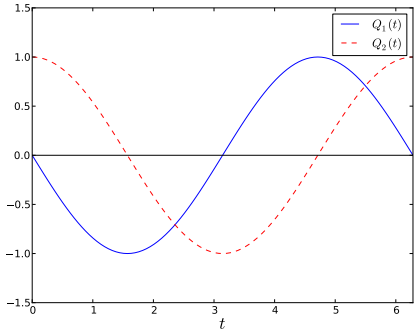
The result is shown in Figure 2a. Finally, we compute the phase interaction function $H(\chi) = H_{12}(\chi) = H_{21}(\chi)$ of two coupled Andronov-Hopf oscillators with coupling:

$$g_{12}(x_1, y_1, x_2, y_2) = x_2 - x_1 = -g_{21}(x_1, y_1, x_2, y_2),$$

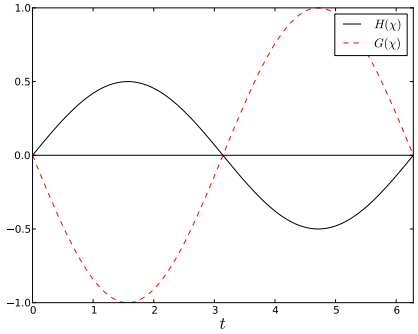
and also the synchronization function $G(\chi) = H(-\chi) - H(\chi)$. Whe show the results in Figure 2b.

```
x1,y1,x2,y2 = var('x1,y1,x2,y2')
g120 = x2-x1
g121 = 0
g12 = [g120, g121]    # Coupling function

aht,H,G = synch(ah,g12,icah,periodah)
```



(A) Components of the solution of the Adjoint system.



(B) Phase interaction and synchronization functions of two coupled Andronov-Hopf oscillators with coupling (g_{12}, g_{21}).

FIG. 2. Results for the Andronov-Hopf oscillator. See text for more details.

4.2. The Selkov model. Our next example will be the so-called Selkov model, which was initially proposed as a model for self-oscillations in glycolysis (see [**27**]), but that has been widely used in models for circadian rhythms (see for instance [**5**] and [**22**]). It is given by the system:

$$\begin{cases} \dot{x} &= 1 - xy, \\ \dot{y} &= ay\dfrac{x-(1+b)}{1+by}, \end{cases} \tag{23}$$

with $a, b \in \mathbb{R}$. We will take the parameters $a = 3$, $b = 1$.

Again, we define the equations:

```
a = 3
b = 1
selk1 = 1-x*y
selk2 = a*y*(x-(1+b)/(1+b*y))
selk = [selk1,selk2]
```

and look for some initial conditions lying in the periodic orbit and the period:

```
ic = [1.0,3.0]
icselk, periodselk = find_po(selk,ic)
```

obtaining:

$$\texttt{icselk} = [1.38276467841, 3.0]\,, \qquad \texttt{periodselk} = 6.34389490962.$$

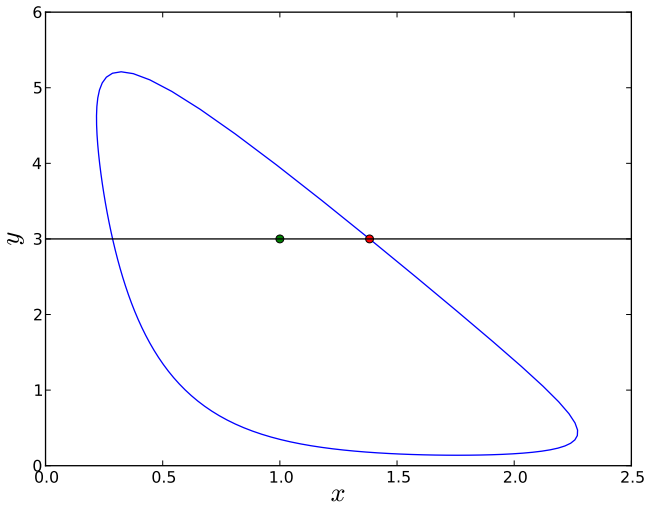


FIG. 3. Periodic orbit of the Selkov model (23). In green, `ic`. In red, `icselk`.

We solve the adjoint system (result shown in Figure 4a):

```
selkt,selkq = adjoint(selk,icselk,periodselk)
```

Finally, we compute the phase interaction function $H(\chi)$ of two Selkov systems with the same interaction function as before, as well as the synchronization function $G(\chi)$ (results shown in Figure 4b):

```
selkt,H,G = synch(selk,g12,icselk,periodselk)
```

4.3. Reduced Hodgkin-Huxley model. Now we will consider a reduced Hodgkin-Huxley-like system, with sodium and potassium currents and only one gating variable:

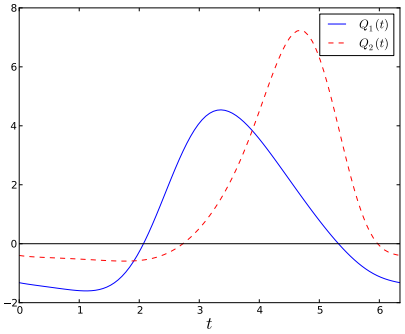
$$\left\{\begin{array}{rcl} \dot{V} &=& -\dfrac{1}{C_m}\left[g_{Na}m_\infty(V)(V-V_{Na})\right. \\ && \left. +g_K n(V-V_k)+g_L(V-V_L)-I_{app}\right], \\ \dot{n} &=& n_\infty(V)-n, \end{array}\right. \tag{24}$$

where:

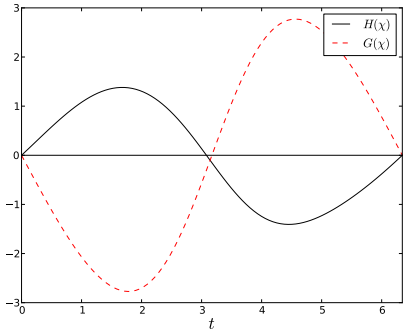
$$m_\infty(V)=\frac{1}{1+\exp(-(V-V_{max,m})/k_m)}, \quad n_\infty(V)=\frac{1}{1+\exp(-(V-V_{max,n})/k_n)}.$$

In the following we reproduce the same steps as before.

```
# Parameters
cm = 1
gna = 20
```



(A) Components of the solution of the Adjoint system.



(B) Phase interaction and synchronization functions for two coupled Selkov systems with coupling $g = (g_{12}, g_{21})$.

FIG. 4. Results for the Selkov model. See text for more details.

```
vna = 60
gk = 10
vk = -90
gl = 8
vl = -80
vmaxm = -20
km = 15
vmaxn = -25
kn = 5
Iapp = 165

# Auxiliar function
mn_inf(v,vmax,k)=1/(1+exp(-(v-vmax)/k))

# Equations
hh1 = -1/cm*(gna*mn_inf(x,vmaxm,km)*(x-vna)+gk*y*(x-vk)+gl*(x-vl)-Iapp)
hh2 = mn_inf(x,vmaxn,kn)-y
hh = [hh1,hh2]

# Initial conditions and period of the periodic orbit
ic = [-15,0.65]
ichh,periodhh = find_po(hh,ic)
```

We obtain:

$$\texttt{ichh} = [\texttt{-6.3675973349}, \texttt{0.65}]\,, \qquad \texttt{periodhh} = \texttt{1.63029898952}.$$

We also compute the solution of the adjoint system, phase interaction function (with the same coupling function as before) and synchronization function. Results are shown in figures 6a and 6b.

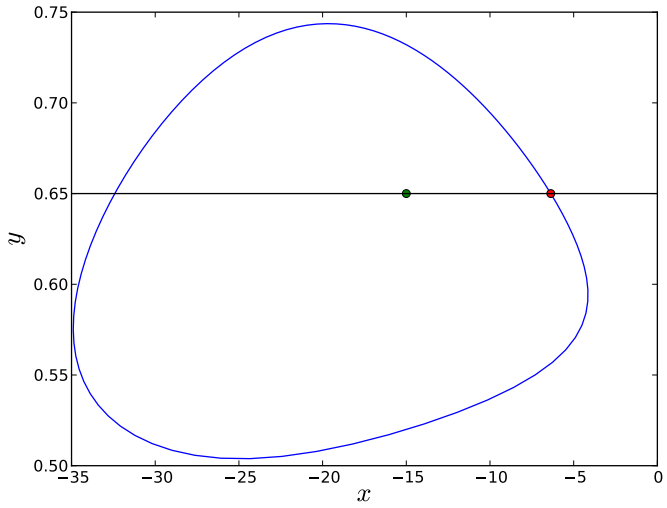
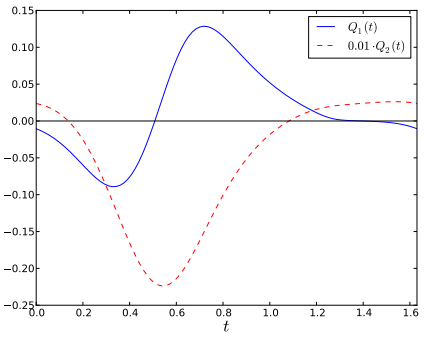


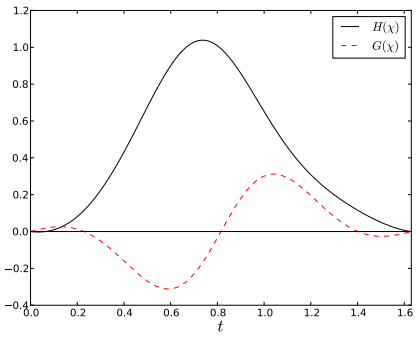
FIG. 5. Periodic orbit of the reduced Hodgkin-Huxley model (24). In green, `ic`. In red, `ichh`.

```
hht,hhq = adjoint(hh,ichh,periodhh)

hht,H,G = synch(hh,g12,ichh,periodhh)
```



(A) Components of the solution of the Adjoint system (rescaled).



(B) Phase interaction and synchronization functions for two coupled reduced Hodgkin-Huxley systems with coupling $g = (g_{12}, g_{21})$.

FIG. 6. Results for the reduced Hodgkin-Huxley model. See text for more details.

4.4. Morris-Lecar model. Our last example will be the Morris-Lecar model (see [**19**]). Originally it was used as a model for a barnacle giant muscle fiber, but it has been studied in the context of neuroscience (see for example [**24**], from where we have taken the values of the parameters). This model is given by the equations:

$$\begin{cases} \dot{V} &= \frac{1}{C}\left[I - g_L(V-V_L) - g_K w(V-V_K) - g_{Ca} m_\infty(V)(V-V_{Ca})\right], \\ \dot{w} &= \phi \frac{w_\infty(V) - w}{\tau_w(V)}, \end{cases} \tag{25}$$

where:

$$\begin{aligned} m_\infty(V) &= \frac{1}{2}\left[1 + \tanh((V-V_1)/V_2)\right], \\ w_\infty(V) &= \frac{1}{2}\left[1 + \tanh((V-V_3)/V_4)\right], \quad \text{and} \\ \tau_w(V) &= (\cosh((V-V_3)/(2V_4)))^{-1}. \end{aligned}$$

Again, we follow the same procedure as before:

```
# Parameters
vl = -60
vk = -84
vca = 120
v1 = -1.2
v2 = 18
v3 = 12
v4 = 17.4
gl = 2
gk = 8.0
gca = 4.0
c = 20
phi = 0.066667
I = 96

# Auxiliar functions
minf(x) = 1/2*(1+tanh((x-v1)/v2))
winf(x) = 1/2*(1+tanh((x-v3)/v4))
tauw(x) = 1/(cosh((x-v3)/(2*v4)))

# Equations
ml1 = 1/c*(I-gl*(x-vl)-gk*y*(x-vk)-gca*minf(x)*(x-vca))
ml2 = phi*(winf(x)-y)/tauw(x)
ml = [ml1,ml2]

# We compute a point of the periodic orbit and its point
ic = [-40,0.3]
icml,periodml = find_po(ml,ic)

# We solve the adjoint system
mlt,mlq = adjoint(ml,icml,periodml)
```

```
# We compute the phase interaction and synchronization
# functions (same coupling as before)
mlt,H,G = synch(ml,g12,icml,periodml)
```

obtaining:

$$\texttt{icml} = [\texttt{-22.5285708717}, \texttt{0.3}]\,, \qquad \texttt{periodml} = \texttt{42.7997521763},$$

and the results captured in figures 7, 8a and 8b

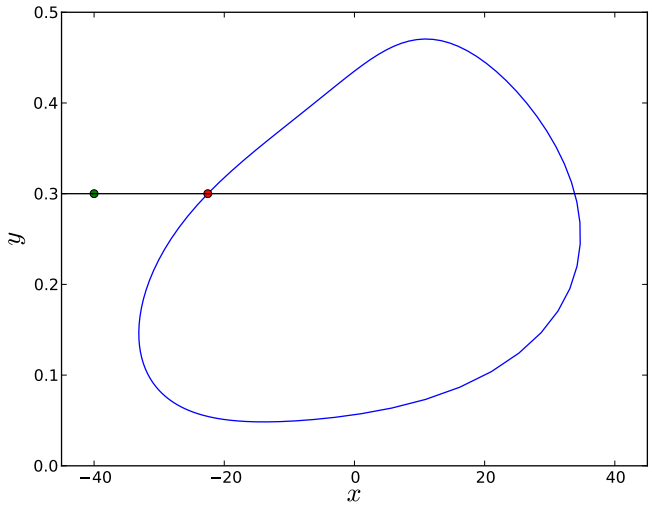
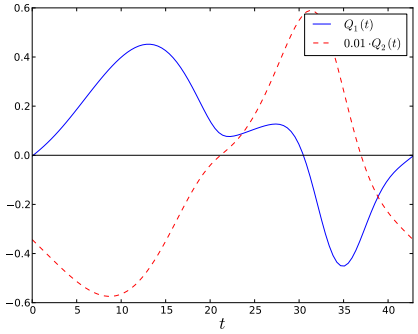


FIG. 7. Periodic orbit of the Morris-Lecar model (25). In green, `ic`. In red, `icml`.

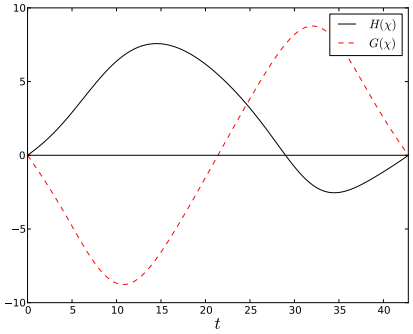
5. Adaptation of `odeint`

As it has been said before, a great feature of Sage is the possibility to combine symbolic manipulation and numerical computations. However, standard routines to solve systems of ODEs given by symbolic functions seemed unsatisfactory to us because of their poor efficiency or by the fact that it took too long to initialize them. For this reason we decided to adapt the routine `odeint` (from scipy.integrate module, a standard Python library). This adaptation, which we called `desolve_odeint`, has been submitted to the Sage Development Team and has been reviewed positively. Therefore, it will be present in the next release of the software and available to all users.

The routine `odeint` approximates numerically systems of first-order differential equations using the solver `lsoda` from the Fortran library `odepack`. We will start by explaining briefly how this routine works, and finally we will explain our adaptation of it to the context of symbolic functions. For more information or details of `lsoda`, we refer the reader to [**15**].



(A) Components of the solution of the Adjoint system (rescaled).



(B) Phase interaction and synchronization functions for two coupled Morris-Lecar systems with coupling $g = (g_{12}, g_{21})$.

FIG. 8. Results for the Morris-Lecar model. See text for more details.

One of the main features of `lsoda` is the automatic detection of stiffness. That is, at each step of integration the routine checks if the problem appears to be stiff or not; in case it is, it uses a Backward Differentiation Formula method to compute the solution, and on the contrary it uses an Adams predictor-corrector method. We now proceed to describe these two methods; we will mainly follow the outline of [**14**].

5.1. The Adams methods. The Adams methods can be divided in two categories: the explicit and the implicit ones. The first kind is computationally less expensive than the latter, but the accuracy is lower. Therefore a combination of the two methods is often made, using a predictor-corrector scheme. This is precisely what `lsoda` does.

5.1.1. *Explicit Adams methods.* Consider a system of first order differential equations:

$$\left\{ \begin{array}{lcl} x' & = & f(t,x), \\ x(t_0) & = & x_0. \end{array} \right. \tag{26}$$

Consider a set of points $t_i = t_0 + ih$, and suppose that an approximation of the solution of (26) is known at k points $t_{n-k+1}, \ldots, t_n$:

$$x_j \approx x(t_j), \qquad j = n-k+1, \ldots, n.$$

We want now to compute the approximated value of $x(t_{n+1})$. Writing (26) in its integral form, we have that:

$$x(t_{n+1}) = x(t_n) + \int_{x_n}^{x_{n+1}} f(s, x(s))ds. \tag{27}$$

The idea now is to compute an approximated value of the integral in this expression. In this method it is done by replacing $f(t, x(t))$ by its interpolating polynomial $p(t)$

through the points $\{(t_i, f_i) \,|\, i = n-k+1, \ldots, n\}$, where $f_i = f(t_i, x_i)$. Defining the backward differences:

$$\nabla^0 f_n = f_n, \qquad \nabla^{j+1} f_n = \nabla^j f_n - \nabla^j f_{n-1},$$

this polynomial can be written as:

$$p(t) = p(t_n + sh) = \sum_{j=0}^{k-1} (-1)^j \binom{-s}{j} \nabla^j f_n. \tag{28}$$

Replacing f in (27) by the interpolating polynomial and using (28) we can approximate the value of $x(t_n)$ by:

$$x_{n+1} = x_n + \int_{x_n}^{x_{n+1}} p(t)dt = x_n + h \sum_{j=0}^{k-1} \gamma_j \nabla^j f_n, \tag{29}$$

where the coefficients γ_j are defined by:

$$\gamma_j = (-1)^j \int_0^1 \binom{-s}{j} ds.$$

REMARK 12. The first values $x_1, \ldots, x_{k-1}$ can be obtained by another method, such as a Runge-Kutta method, or by an Adams method of lower order. This is also valid for the Implicit Adams methods and BDF methods.

REMARK 13. The formulas (29) require the interpolating polynomial to be integrated from t_n to t_{n+1}, i.e. outside the interpolation interval $[t_{n-k+1}, t_n]$. It is known that the approximation of the interpolation polynomial outside this interval is in general quite bad; the next type of Adams methods solves this problem.

5.1.2. *Implicit Adams methods.* If we want the interpolation interval to contain x_{n+1}, the only solution is to add the point (t_{n+1}, f_{n+1}) to the interpolating polynomial, that is:

$$p^*(t) = p^*(t_n + sh) = \sum_{j=0}^{k} (-1)^j \binom{-s+1}{j} \nabla^j f_{n+1}.$$

Repeating the same procedure as before with $p^*(t)$ we obtain the formulas:

$$x_{n+1} = x_n + h \sum_{j=0}^{k} \gamma_j^* \nabla^j f_{n+1}, \tag{30}$$

where the γ_j^* are defined as follows:

$$\gamma_j = (-1)^j \int_0^1 \binom{-s+1}{j} ds.$$

REMARK 14. Note that x_{n+1} in (30) is defined implicitly, because the value f_{n+1} depends on it. Hence, the price we pay to obtain a more accurate approximation is having to solve an equation (in general, nonlinear) at each step. Once again, the next method we present gives a solution to this problem.

5.1.3. *Predictor-corrector methods.* A way of solving the nonlinear equation (30) is to compute successive approximations of x_{n+1} and f_{n+1}. The procedure would be for example:

(1) Prediction: compute a first approximation by the explicit Adams method:
$$\hat{x}_{n+1} = x_n + h\sum_{j=0}^{k-1} \gamma_j \nabla^j f_n.$$
(2) Evaluation: evaluate the function at this approximation:
$$\hat{f}_{n+1} = f(t_{n+1}, \hat{x}_{n+1}).$$
(3) Correction: compute a new approximation of x_{n+1} with the implicit Adams method, using $\hat{f}_{n+1}$:
$$x_{n+1} = x_n + h\sum_{j=0}^{k} \gamma_j^* \nabla^j \hat{f}_{n+1}.$$
(4) Evaluation: evaluate the function at this new approximation, obtaining the value of f_{n+1} to be used in the next step.

This is the most used procedure, and it is denoted by PECE. However, more corrections and evaluations per step can be done (like the PECECE) or the last evaluation can be avoided (like the PEC, which uses $\hat{f}_{n+1}$ in the next step instead of f_{n+1}).

5.2. The Backward Differentiation Formulas (BDF). The starting point of BDF methods is the same as for the Adams methods: considering $x_{n-k+1}, \ldots, x_n$, the approximations of (26) at points $t_{n-k+1}, \ldots, t_n$. However, now we take the interpolation polynomial through the points $\{(t_i, x_i) \,|\, i = n-k+1, \ldots, n+1\}$ (and not through (t_i, f_i) as before), which can be written as:
$$q(t) = q(t_n + sh) = \sum_{j=0}^{k} (-1)^j \binom{-s+1}{j} \nabla^j x_{n+1}.$$
Note that this expression depends on x_{n+1}, which is to be determined. We can take x_{n+1} such that $q(t)$ satisfies the equation (26) at t_{n+1}, i.e. imposing that:
$$q'(t_{n+1}) = f(t_{n+1}, x_{n+1}).$$
Thus we obtain the implicit formula:
$$\sum_{j=0}^{k} \delta_j^* \nabla^j x_{n+1} = h f_{n+1}, \tag{31}$$
where:
$$\delta_j^* = (-1)^j \frac{d}{ds} \binom{-s+1}{j}\Big|_{s=1}.$$
Differentiating the expression for of the binomial coefficient:
$$(-1)^j \binom{-s+1}{j} = \frac{1}{j!}(s-1)s(s+1)\cdots(s+j-2)$$

the coefficients d_j^* can be obtained easily:

$$d_0^* = 0, \qquad d_j^* = \frac{1}{j}, \quad j \geq 1.$$

Then the formula (31) becomes:

$$\sum_{j=0}^{k} \frac{1}{j} \nabla^j x_{n+1} = h f_{n+1}.$$

This implicit equation can be solved by Newton's method, but the Jacobian matrix of f is needed.

REMARK 15. The BDF methods are stable up to order $k = 6$.

5.3. The `desolve_odeint` routine. We proceed now to explain how is our routine structured. First of all, we briefly describe its main parameters (check the examples to see how they are used):

- **des**: the field of the system, given by a symbolic expression.
- **ics**: the initial conditions.
- **times**: a list of all the points in which the solution must be found.
- **dvars**: a list of the dependent variables.
- **ivar** (optional): the dependent variable.
- **compute_jac** (optional): A boolean; in case that it is True, the Jacobian of the field is computed symbollicaly. Then, if the system is stiff, it is used by the BDF method. If **compute_jac** is False and the system is stiff, the Jacobian is approximated by `odeint` itself (by difference quotients). Default value is False.

Some other parameters can be specified, such as the absolute and relative tolerances; we refer the reader to the examples to see how they are used.

The routine `desolve_odeint` has been under a refereeing process, from which many different versions have arisen. The final version, which has been accepted to be part of Sage 4.6, has mainly three parts:

- First of all, if the independent variable is not provided, the routine tries to guess which one it is. In case the system is autonomous, which means that the independent variable does not appear in any equation, a new variable is created.
- In the second part, the symbolic functions of **des** are compiled, to make the evaluation much faster and consequently the numerical integration much more efficient. If **compute_jac** is set to True, the Jacobian is computed symbolically and then compiled too.
- Finally, the routine `odeint` is called, providing the numerical solution of the system.

For more detail, we refer the reader to the Appendix A.7.

5.4. Examples. Now we proceed to present some examples to illustrate the use of the routine `desolve_odeint` and its optional parameters.

5.4.1. *Lotka-Volterra Equations.* Our first example will be the Lotka-Volterra equations, which are one of the most famous models in mathematical biology and represent the interaction of two species, one being the predator and the other its prey. These equations are:

$$\begin{cases} x' &= x(\alpha - \beta y), \\ y' &= -y(\gamma - \delta x). \end{cases} \tag{32}$$

We will take the parameters $\alpha = \beta = \gamma = \delta = 1$.

First of all, we have to declare the symbolic variables we are going to use and the introduce then equations:

```
# Declare the symbolic variables
x,y = var('x,y')

# Lotka-Volterra Equations
lveqs = [x*(1-y),-y*(1-x)]
```

Then, we choose some initial conditions:

```
ics = [4.2,2.8]
```

and a list of times at which the solution must be computed:

```
times = srange(0,30,0.01) # Time goes from 0 to 30 with step 0.01
```

Finally, we call `desolve_odeint`:

```
sol=desolve_odeint(lveqs,ics,times,[x,y])
```

We can plot the solution (y versus x) by introducing:

```
line(zip(sol[:,0],sol[:,1]))
```

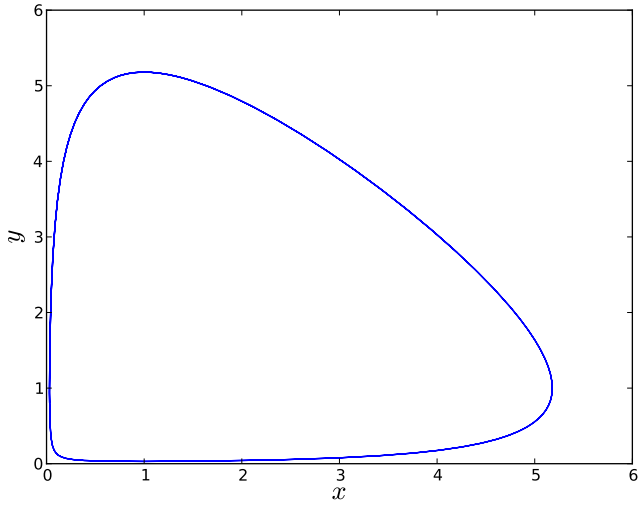


FIG. 9. Numerical solution of the Lotka-Volterra equations (32) found with `desolve_odeint`.

Or similarly plot both variables versus time:

```
line(zip(times,sol[:,0]),color='green')+line(zip(times,sol[:,1]),color='red')
```

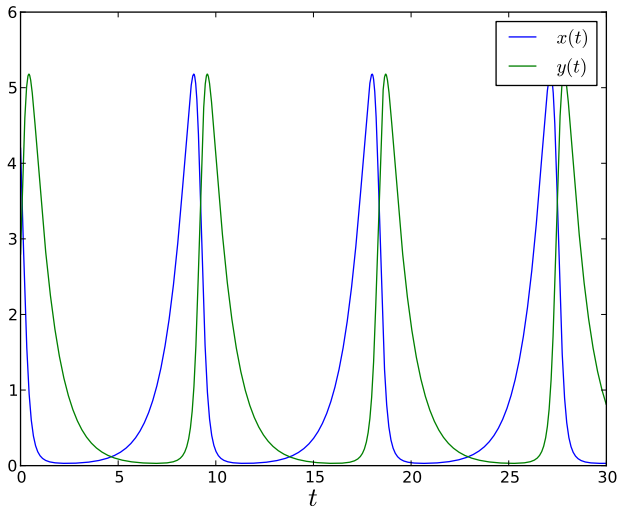


FIG. 10. Both components of the solution of the Lotka-Volterra equations (32) found with `desolve_odeint`.

5.4.2. *A simple combustion model.* Our next example is a one-dimensional combustion model which is stiff (see [**23**]), so that we will use the argument `compute_jac`. The equation is:

$$y' = y^2(1-y), \quad y(0) = \varepsilon, \tag{33}$$

where $y(t) \geq 0$ represents the concentration of the reacting chemical at time t. Again, first of all we define the variable, parameter and function:

```
epsilon = 0.01
f = y^2*(1-y)
```

We define the initial condition, and the list of times:

```
ic = epsilon
time = srange(0,2/epsilon,1)
```

Now we will call the solver setting the argument `compute_jac` to True, and specifying the relative and absolute tolerances:

```
sol = desolve_odeint(f,ic,time,y,rtol=1e-9,atol=1e-10,compute_jac=True)
```

Finally, we plot the solution with points:

```
points(zip(time,sol))
```

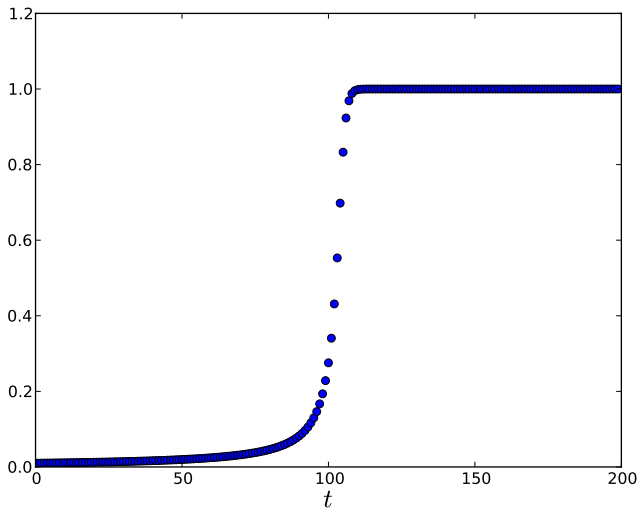


FIG. 11. Solution of the combustion model (33) found with `desolve_odeint`.

5.4.3. *A chemical reaction.* Finally, we present another stiff differential equation, whose solutions change rapidly over many orders of magnitude, which we have taken

from [**14**]. It models the chemical reaction between three different substances and it is given by:

$$\begin{cases} y_1' &= 77.27\left[y_2 + y_1\left(1 - 8.375 \cdot 10^{-6} y_1 - y_2\right)\right], \\ y_2' &= \dfrac{1}{77.27}\left[y_3 - (1 + y_1)\, y_2\right], \\ y_3' &= 0.161(y_1 - y_3). \end{cases} \tag{34}$$

We proceed as the previous examples:

```
y1,y2,y3 = var('y1,y2,y3')
f1 = 77.27*(y2+y1*(1-8.375*1e-6*y1-y2))
f2 = 1/77.27*(y3-(1+y1)*y2)
f3 = 0.16*(y1-y3)
f = [f1,f2,f3]
ci = [1,1,1]
t = srange(0,400,0.05)
v = [y1,y2,y3]
```

Finally, we call `desolve_odeint`, specifying the following optional parameters:

- The relative and absolute tolerances (`rtol` and `atol`).
- The step size to be attempted on the first step (`h0`).
- The maximum and minimum absolute step sizes allowed (`hmax` and `hmin` respectively).
- Maximum number of (internally defined) steps allowed for each integration point in t (`mxstep`).
- Maximum order to be allowed for the stiff (BDF) method (`mxords`).

with the line:

```
sol = desolve_odeint(f,ci,t,v,rtol=1e-3,atol=1e-4,h0=0.1,
hmax=1,hmin=1e-4,mxstep=1000,mxords=4)
```

Note that in this case `desolve_odeint` has its default value (that is, False) and therefore the Jacobian matrix of `f` will be approximated by finite differences.

6. The routine `create_odesolver`

The routine `desolve_odeint` provides both an efficient and a simple way to solve systems of ODEs, but yet it has some aspects, inherent to the routine `odeint`, that can be improved. First of all, it can be somewhat inconvenient the fact that it is impossible to choose the numerical method to be used to integrate the system. On the other hand, if the integration must be repeated many times, for example each time with a different initial condition, the symbolic function must be also compiled each time, which results in a significant loss of effectiveness.

For these reasons, we created a routine that initializes a Sage class, namely the class `ode_solver` (which wraps the GSL libraries to solve systems of ODEs, see [**9**] for more information), and makes it compatible with symbolic functions. The

fact that we now create a class solves the problem of having to re-compile the symbolic functions: it is only done when the class is initialized, and afterwards all the parameters can be changed as many times as wanted without having to do it all over again. Besides, this class provides the possibility of choosing among some numerical methods, solving the other inconvenient of `desolve_odeint` too. These methods, which we will only mention, are:

- Runge-Kutta-Fehlberg (4,5)
- Embedded Runge-Kutta (2,3)
- 4th order classical Runge-Kutta
- Runge-Kutta Prince-Dormand (8,9)
- Implicit 2nd order Runge-Kutta at Gaussian points
- Implicit 4th order Runge-Kutta at Gaussian points
- Implicit Bulirsch-Stoer (this method requires the Jacobian, but our routine will use the symbolic capabilities of Sage to compute it).
- M=1 implicit Gear
- M=2 implicit Gear

The main parameters of `create_odesolver` are quite similiar to these of `desolve_odeint`:

- **f**: the field of the system, given by a symbolic expression.
- **y_0**: the initial conditions.
- **dvars**: a list of the dependent variables.
- **ivar** (optional): the dependent variable. If it is not provided, it is set to `t`.
- **algorithm** (optional): The method to be used. Its default value is the Runge-Kutta-Fehlberg (4,5).

The absolute and relative tolerances, and other parameters, can also be specified. Again, we refer the reader to the examples for more information on them.

Unlike `desolve_odeint`, the routine `create_odesolver` does not solve the system, it just initializes the class. It has two main parts:

- First of all, the function **f** is compiled. In case the algorithm parameter is set to `"bsimp"`, that is the implicit Bulirsch-Stoer method, the Jacobian of the function is symbolically computed and then compiled too.
- The class `ode_solver` is initialized with the corresponding parameters.

In the following, we will present some examples to illustrate the use of this routine. For more details on its structure, we refer the reader to Appendix A.8.

The routine `create_odesolver` has not yet been submitted to Sage developers, but we plan to do it briefly.

6.1. Examples. For the illustration of the routine `create_odesolver` we will take two classical dynamical systems.

6.1.1. *The van der Pol Oscillator.* First of all we will consider the van der Pol oscillator, written in the form:

$$\begin{cases} x' &= y, \\ y' &= \varepsilon(1-x^2)y - x. \end{cases} \tag{35}$$

The first steps, namely the initialization of the parameter, variables, equations and initial conditions, are exactly the same as we did wit the routine `desolve_odeint`:

```
x,y=var('x,y')
epsilon=0.01
vdp=[y,epsilon*(1-x^2)*y-x]
ics=[1.5,0.7]
```

Now we call `create_odesolver`:

```
T=create_odesolver(vdp,[x,y],y_0=ics)
```

which initializes an element of the class `ode_solver`. Note that the initial conditions are an optional argument, because they are not necessary for the initialization. Finally, we solve the equations by introducing:

```
T.ode_solve(t_span=[0,10], num_points=100)
```

Here the interval on which to solve the ODE, `t_span`, must be specified. In case it is a tuple with just two time values, the user must specify `num_points`, and the system will be evaluated at this number of points equally spaced between `t_span[0]` and `t_span[1]`.

If we want to plot one component of the solution, we can do it by writing:

```
T.plot_solution(i=0) # We plot the first component
```

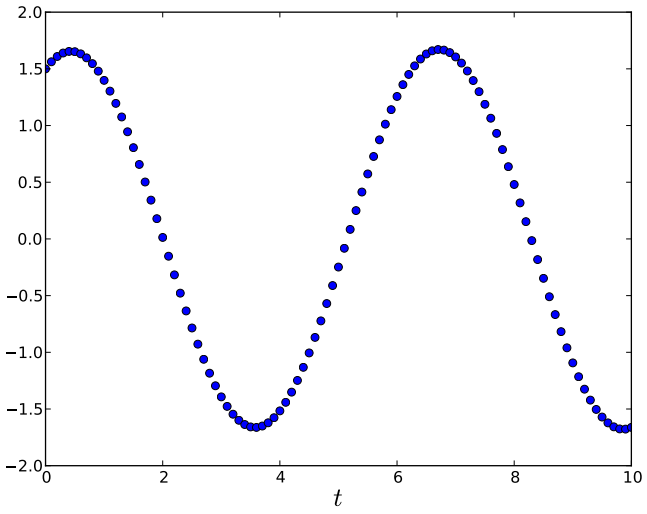


FIG. 12. Solution of the van der Pol equations (35) found with `create_odersolver`.

6.1.2. *The Lorenz Equations.* Now consider the Lorenz equations:

$$\left\{\begin{array}{rcl} x' &=& \sigma(y-x), \\ y' &=& x(\rho-z)-y, \\ z' &=& xy-\beta z, \end{array}\right. \tag{36}$$

with parameter values $\sigma = 10$, $\rho = 28$ and $\beta = 8/3$. We begin as usual:

```
# We declare the variables
x,y,z = var('x,y,z')

# We define the parameters
sigma = 10
rho = 28
beta = 8/3

# The Lorenz equations
lorenz = [sigma*(y-x),x*(rho-z)-y,x*y-beta*z]

# Time and initial conditions
times = srange(0,50.05,0.005)
ics = [0,1,1]
```

Now we call our routine, specifying the relative and absolute tolerances (`error_rel` and `abs_rel`) and using the algorithm Runge-Kutta Prince-Dormand (8,9), and solve the equation:

```
T = create_odesolver(lorenz,[x,y,z],y_0=ics,error_rel=1e-9,
```

```
    error_abs = 1e-9,algorithm = 'rk8pd')
    T.ode_solve(t_span = times)
```

Note that in this case, `t_span` is a list with more than two points, and therefore `num_points` is not needed.

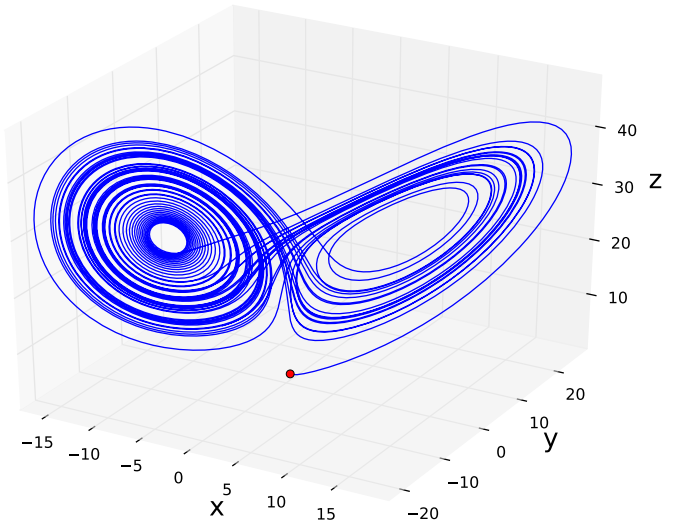


FIG. 13. Numerical solution of the Lorenz equations (36) found with `create_odesolver`. In red, the initial point `ics`.

7. The Brian package

Finally, we will present the Sage package of Brian. It has been under a refereeing process and finally it has been accepted as an experimental package, but probably in the future it will be an optional package. It can be downloaded from the Sage web page: `www.sagemath.org/packages/experimental`.

The structure of this section will be the following: first of all, we will give a short description of the main features of Brian. Then we will explain how the creation of the Sage package was done and its structure, and finally we will give some examples of its use.

7.1. Main features. Brian, as we already mentioned before, is a simulator for spiking neural networks. In this sense, its main goal (which is the same as what we had in mind in the previous sections) is to provide a platform that is both efficient (and here efficient means not much slower than a compiled language like C) and simple to use. Thus, it is designed to solve the problems that appear often in neuroscience. The most remarkable features are the following:

- Equations: These are the equations that model the behavior of the neurons. There are four types of equations: differential equations, (non-differential) equations, aliases (where 2 variables are set to be equal) and parameters. Equations are initialized as follows:

```
eqs=Equations('''
dx/dt=(y-x)/second + a : volt   # differential equation
y=2*x : volt                    # equation
z=x                             # alias
a : volt/second                 # parameter
''')
```

- Neuron groups: Once one has the equations, a group of neurons can be defined with the command `NeuronGroup`. For example, to define a group of 100 neurons modeled by the equations `eqs` we should write:

```
G=NeuronGroup(100,model=eqs)
```

  Reset and threshold can be specified for integrate-and-fire models (in which the membrane potential is reset to some value when it reaches a given maximal voltage). Subgroups of a given group can be defined simply with the following code:

```
S1=G.subgroup(30)
S2=G.subgroup(70)
```

- Connections: Different types of connections between the neurons of a group can be specified. Although custom connections can be defined, there are some connectivity functions that are already predefined, such as `connect_full`, `connect_random`, etc.
- Units: Quantities with physical units can be defined, and operations between them can be done. The fundamental and derived SI units can be used.
- Monitors: Spikes and state variables can be easily monitored.
- Integration methods: there is exact integration for linear models, and Euler, Runge-Kutta and exponential Euler methods for nonlinear models. Stochastic differential equations are also possible.

7.2. Package structure. To create the package, our task was mainly to set all the code written by the developers of Brian (see [**10**] and [**11**]) according to the structure of Sage packages and creating a patch to avoid some minor errors.

We proceed now to describe the structure that Sage packages must have. The first thing one has to keep in mind is that all the files must be inside a directory whose name contains the package name and version; it will be later compressed with Sage in a `.spkg` file and will be the actual package. This directory must contain the following items:

- The `src/` directory, which contains an unmodified version of the software. In our case, this directory contains all the code of the original Brian package.
- The `patches/` directory, which contains all the patches that have to be applied to the code in `src/` during the installation. In our case, there is just a small patch (which we created with the help of Brian developers) to avoid problems between Brian units and Sage.

- The installation script `spkg-install`. Prior to installing the package, this script must check that all the other packages on which this package depends are already installed, and then apply the patches specified in the `patches/` directory.
- The `SPKG.txt` file, where a general description of the package is provided, as also its license, maintainers, dependencies, and other specific information.

The Sage project uses Mercurial for its revision control system (see [**20**] for a survey, or just its web page `http://mercurial.selenic.com`). In order to make the revision control possible, two more (hidden) items must be added in the directory:

- The `.hgignore` file, which specifies the files or folders that should not be tracked by Mercurial. It must contain the `src/` directory, because the code in it is not modified.
- The `.hg` directory, that contains the Mercurial repository for all files not in the `src/` directory.

7.3. Examples.

7.3.1. *A Hodkgin-Huxley network.* Our first example will be a network of 500 Hodgkin-Huxley-like neurons. First of all, we must import Brian and declare the parameters:

```
from brian import *

# Parameters
area = 20000*umetre**2
Cm = (1*ufarad*cm**-2)*area
gl = (5e-5*siemens*cm**-2)*area
El = -60*mV
EK = -90*mV
ENa = 50*mV
g_na = (100*msiemens*cm**-2)*area
g_kd = (30*msiemens*cm**-2)*area
VT = -63*mV

# Time constants
taue = 5*ms
taui = 10*ms

# Reversal potentials
Ee = 0*mV
Ei = -80*mV
we = 6*nS # excitatory synaptic weight (voltage)
wi = 67*nS # inhibitory synaptic weight
```

Next, we define the equations of the Hodgkin-Huxley model:

```
# The model
eqs=Equations('''
dv/dt = (gl*(El-v)+ge*(Ee-v)+gi*(Ei-v)-
```

```
g_na*(m*m*m)*h*(v-ENa)-g_kd*(n*n*n*n)*(v-EK))/Cm : volt
dm/dt = alpham*(1-m)-betam*m : 1
dn/dt = alphan*(1-n)-betan*n : 1
dh/dt = alphah*(1-h)-betah*h : 1
dge/dt = -ge*(1./taue) : siemens
dgi/dt = -gi*(1./taui) : siemens
alpham = 0.32*(mV**-1)*(13*mV-v+VT)/(exp((13*mV-v+VT)/(4*mV))-1.)/ms:Hz
betam = 0.28*(mV**-1)*(v-VT-40*mV)/(exp((v-VT-40*mV)/(5*mV))-1)/ms:Hz
alphah = 0.128*exp((17*mV-v+VT)/(18*mV))/ms : Hz
betah = 4./(1+exp((40*mV-v+VT)/(5*mV)))/ms : Hz
alphan =0.032*(mV**-1)*(15*mV-v+VT)/(exp((15*mV-v+VT)/(5*mV))-1.)/ms:Hz
betan = .5*exp((10*mV-v+VT)/(40*mV))/ms : Hz
''')
```

Then we define a group of 500 neurons, specifying a threshold (more concretely, an empirical threshold, which is special for Hodgkin-Huxley models). We will also use an implicit integration method (setting the `implicit` argument to True), because Hodgkin-Huxley models are stiff:

```
P=NeuronGroup(500,model=eqs,
threshold=EmpiricalThreshold(threshold=-20*mV,refractory=3*ms),
implicit=True)
```

Now we define two subgroups, one of inhibitory neurons and the other one of excitatory, which are randomly connected to the whole group `P` through the variables `ge` and `gi` respectively, with synaptic weights of `we` and `wi`:

```
Pe=P.subgroup(350)
Pi=P.subgroup(150)
Ce=Connection(Pe,P,'ge',weight=we,sparseness=0.02)
Ci=Connection(Pi,P,'gi',weight=wi,sparseness=0.02)
```

We initialize the variables `v`, `ge` and `gi` of each neuron in `P` with some random values:

```
P.v=El+(randn(len(P))*5-5)*mV
P.ge=(randn(len(P))*1.5+4)*10.*nS
P.gi=(randn(len(P))*12+20)*10.*nS
```

Now we declare a state monitor, which will record the state variable `v` (the membrane potential) of neurons number 1, 10 and 100:

```
trace=StateMonitor(P,'v',record=[1,10,100])
```

Finally, we run the network, and plot the result (shown in Figure 14):

```
run(500*msecond)
plot(trace.times,trace[1],label='Neuron 1')
plot(trace.times,trace[10],label='Neuron 10')
plot(trace.times,trace[100],label='Neuron 100')
legend()
```

```
savefig('plots/hodgkinhuxley.pdf')
```

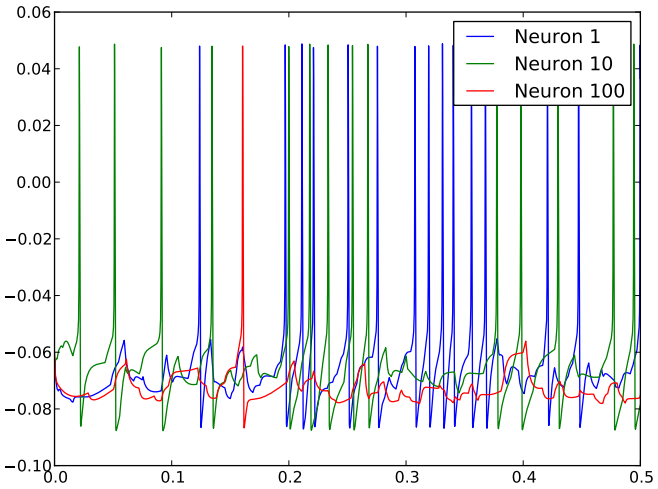


FIG. 14. Simulation of the Hodgkin-Huxley network.

7.3.2. *A raster plot.* Now we will show how to create a raster plot, that is, a graphic that displays the neurons which are firing at each moment. We begin importing Brian and defining the model:

```
from brian import *
eqs = '''
dv/dt = (ge+gi-(v+49*mV))/(20*ms) : volt
dge/dt = -ge/(5*ms) : volt
dgi/dt = -gi/(10*ms) : volt
'''
```

We define a group of 2000 neurons, with threshold at -50 mV and reset at -60 mV, and two subgroups:

```
R = NeuronGroup(2000, eqs, threshold=-50*mV, reset=-60*mV)
Re = R.subgroup(1600)
Ri = R.subgroup(400)
```

We initialize the variable `v` and connect the two subgroups randomly:

```
R.v = -60*mV
Ce = Connection(Re, R, 'ge', weight=1.62*mV, sparseness=0.02)
Ci = Connection(Ri, R, 'gi', weight=-9*mV, sparseness=0.02)
```

We declare a spike monitor, which will record the spikes of our group of neurons, and run the network:

```
M = SpikeMonitor(R)
run(0.5*second)
```

Finally, we create the raster plot (shown in Figure 15):

```
raster_plot(M)
savefig('plots/rasterplot.pdf')
```

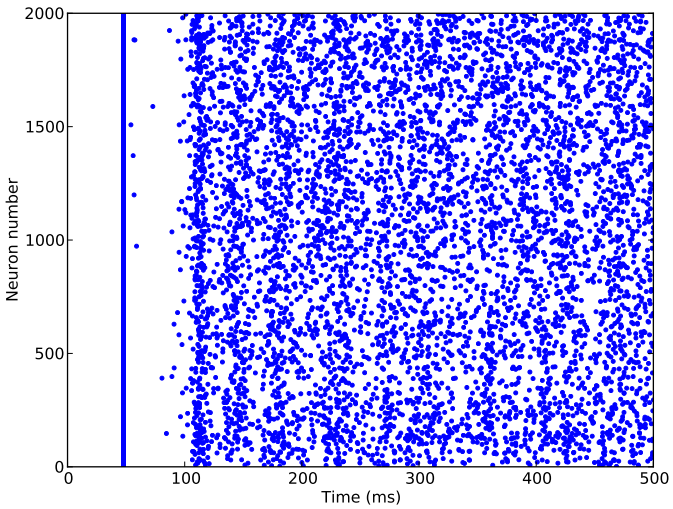


FIG. 15. Raster plot.

Chapter 3
Second order PRCs

In this last chapter we will introduce the second order PRCs, which are, as their name indicates, a generalization of classical PRCs to the second order in the perturbation parameter. It is quite peculiar that in the literature of PRCs, and more generally in mathematical neuroscience, the possibility of studying higher order terms of the phase equations is barely mentioned. However, there has been recent work on higher order local approximations of isochrons (see [**30**]). For these reasons we wanted to explore a little bit what would occur if we went a little bit further and try to obtain a second order term in the case of PRCs, and study in which cases it should be taken into account.

First of all we started with the simplest case, the pulsed coupling, and we obtained a method (equivalent to that presented in [**30**] in the case of order two, but less algebraic and easier to compute) to compute numerically these second order PRCs. In the case of non-pulsed couplings, we realized that the classic framework in which PRCs are derived is no longer valid if we want to consider higher orders, and we need to reformulate it in terms of Poincaré-Lindstedt series.

We will begin by presenting the results we obtained for pulsed couplings. Afterwards we will consider weakly perturbed systems, which need this new framework to be introduced. Finally, we will study weakly connected networks, which are in fact quite similar to the previous ones, but still there are some differences caused by the fact that they involve more than one oscillator.

1. Pulsed couplings

Let us return to the systems of the form:

$$\dot{x} = f(x) + v\delta(t - t_s),$$

where the system $\dot{x} = f(x)$ has a limit cycle $\gamma(t)$. Recall that the instantaneous perturbation $v\delta(t-t_s)$ causes a change of the state variables at time t_s, and our aim was to compute the phase difference between the solution of the unperturbed system at time t_s, that is $x_{t_s} = \gamma(t_s)$, and that of the perturbed system, i.e. $x_{t_s} + v$. In this section we will present a method to compute a higher term of the PRC. Expanding

the difference between the phases at $x_{t_s}+v$ and x_{t_s} up to order two, we have:

$$\vartheta(x_{t_s}+v)-\vartheta(x_{t_s})=\nabla\vartheta(x_{t_s})\cdot v+v^T D\nabla\vartheta(x_{t_s})v+O(|v|^3).$$

We will now present a method to compute $D\nabla\vartheta(\gamma(t))$. With the following lemmas, we will be able to find a differential equation that determines it.

LEMMA 1. *Let $(x(t),y(t))$ be a solution of:*

$$\begin{aligned}\dot{x} &= f_1(x,y),\\ \dot{y} &= f_2(x,y),\end{aligned}$$

where $f=(f_1,f_2)^T:\mathbb{R}^2\to\mathbb{R}^2$ is C^r, $r\geq 2$. Let $Q:\mathbb{R}^2\to\mathbb{R}^2$ a $\mathcal{C}^2$ vector field, and DQ its derivative. Then:

$$\frac{d}{dt}\left[DQ(x(t),y(t))\right]=D\left[\frac{dQ}{dt}(x(t),y(t))\right]-DQ(x(t),y(t))Df(x(t),y(t)). \tag{37}$$

PROOF. First let us compute the left-hand side of the equality:

$$\frac{d}{dt}\left[DQ(x(t),y(t))\right]=\frac{\partial DQ}{\partial x}\dot{x}+\frac{\partial DQ}{\partial y}\dot{y}=\begin{pmatrix}\dfrac{\partial^2 Q}{\partial x^2}f_1+\dfrac{\partial^2 Q}{\partial x\partial y}f_2\\ \dfrac{\partial^2 Q}{\partial x\partial y}f_1+\dfrac{\partial^2 Q}{\partial y^2}f_2\end{pmatrix}=(D^2Q)f \tag{38}$$

On the other hand we have:

$$\begin{aligned}D\left[\frac{dQ}{dt}(x(t),y(t))\right]&=D\left[\frac{\partial Q}{\partial x}\dot{x}+\frac{\partial Q}{\partial y}\dot{y}\right]=D\left[\frac{\partial Q}{\partial x}f_1+\frac{\partial Q}{\partial y}f_2\right]=\\ &=\begin{pmatrix}\dfrac{\partial^2 Q}{\partial x^2}f_1+\dfrac{\partial Q}{\partial x}\dfrac{\partial f_1}{\partial x}+\dfrac{\partial^2 Q}{\partial x\partial y}f_2+\dfrac{\partial Q}{\partial y}\dfrac{\partial f_2}{\partial x}\\ \dfrac{\partial^2 Q}{\partial x\partial y}f_1+\dfrac{\partial Q}{\partial x}\dfrac{\partial f_1}{\partial y}+\dfrac{\partial^2 Q}{\partial y^2}f_2+\dfrac{\partial Q}{\partial y}\dfrac{\partial f_2}{\partial y}\end{pmatrix}=(D^2Q)f+DQDf\end{aligned} \tag{39}$$

From (38) and (39), statement (37) is clear. □

LEMMA 2. *Let $M:\mathbb{R}^2\to\mathcal{M}_{2\times 2}(\mathbb{R})$, and $Q:\mathbb{R}^2\to\mathbb{R}^2$, both continuously differentiable. Then:*

$$D(MQ)=\left[\frac{\partial M}{\partial x}Q\,\middle|\,\frac{\partial M}{\partial y}Q\right]+MDQ$$

PROOF. This is a straightforward computation. If we write:

$$M(x,y)=\begin{pmatrix}m_{11}(x,y) & m_{12}(x,y)\\ m_{21}(x,y) & m_{22}(x,y)\end{pmatrix},$$

and:

$$Q(x,y)=\begin{pmatrix}q_1(x,y)\\ q_2(x,y)\end{pmatrix},$$

we have that:

$$MQ=\begin{pmatrix}m_{11}q_1+m_{12}q_2\\ m_{21}q_1+m_{22}q_2\end{pmatrix}.$$

Differentiating, we obtain:

$$D(MQ) =$$
$$= \begin{pmatrix} \frac{\partial m_{11}}{\partial x} q_1 + m_{11} \frac{\partial q_1}{\partial x} + \frac{\partial m_{12}}{\partial x} q_2 + m_{12} \frac{\partial q_2}{\partial x} & \frac{\partial m_{11}}{\partial y} q_1 + m_{11} \frac{\partial q_1}{\partial y} + \frac{\partial m_{12}}{\partial y} q_2 + m_{12} \frac{\partial q_2}{\partial y} \\ \frac{\partial m_{21}}{\partial x} q_1 + m_{21} \frac{\partial q_1}{\partial x} + \frac{\partial m_{22}}{\partial x} q_2 + m_{22} \frac{\partial q_2}{\partial x} & \frac{\partial m_{21}}{\partial y} q_1 + m_{21} \frac{\partial q_1}{\partial y} + \frac{\partial m_{22}}{\partial y} q_2 + m_{22} \frac{\partial q_2}{\partial y} \end{pmatrix} =$$
$$= \left[\frac{\partial M}{\partial x} Q \middle| \frac{\partial M}{\partial y} Q \right] + MDQ.$$

□

Now we can enunciate the following proposition:

PROPOSITION 1. *If x_0 is a periodic solution of the system:*

$$x' = f(x), \tag{40}$$

and $\nabla\vartheta(x_0)$ is the solution of the corresponding adjoint system, then the derivative of $\nabla\vartheta$ along this periodic solution, $D\nabla\vartheta(x_0)$, is a matrix solution of the system:

$$\dot{U} = -\left[\frac{\partial Df^T(x_0)}{\partial x} Q(t) \middle| \frac{\partial Df^T(x_0)}{\partial x} Q(t) \right] - Df^T(x_0)U - UDf^T(x_0), \tag{41}$$

satisfying:

(1) $U(t)$ is T-periodic,
(2) $Q^T(t)Df(x_0(t)) + f^T(x_0(t))U(t) = \begin{pmatrix} 0 & 0 \end{pmatrix}$,

where $Q(t) = \nabla\vartheta(x_0(t))$.

PROOF. Equation (41) is clear by Lemmas 1 and 2 and recalling that Q satisfies the adjoint system:

$$\dot{Q} = -Df^T(x_0)Q.$$

On the other hand, property 1 is clear since Q is T-periodic, and hence $DQ = D\nabla\vartheta(x_0)$ must be so too. Finally, to prove property 2 recall that:

$$f^T(x_0)\nabla\vartheta(x_0) = 1,$$

and therefore:

$$\begin{pmatrix} 0 & 0 \end{pmatrix} = D\left[f^T(x_0(t))\nabla\vartheta(x_0(t))\right] = \nabla\vartheta^T(x_0(t))Df(x_0(t)) + f^T(x_0(t))D\nabla\vartheta(x_0(t)) =$$
$$= Q^T(t)Df(x_0(t)) + f^T(x_0(t))U(t).$$

□

We will call (41) the *second adjoint equation* of (40) and in the following we will tackle the problem of its computation.

1.1. Numerical computation of $D\nabla\vartheta$. From Proposition 1 we can obtain a method, in a way *analogous* to the Adjoint method, to compute the derivative of $\nabla\vartheta$ along the periodic solution $x_0(t)$. However, first of all we have to re-write this equation as a linear system of ODEs.

We begin by introducing some notation to simplify the process. Let:

$$U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix},$$

$$M(t) = -Df^T(x_0(t)) = \begin{pmatrix} m_{11}(t) & m_{12}(t) \\ m_{21}(t) & m_{22}(t) \end{pmatrix},$$

$$B(t) = -\left[\frac{\partial Df^T(x_0(t))}{\partial x}Q(t) \,\middle|\, \frac{\partial Df^T(x_0(t))}{\partial x}Q(t)\right] = \begin{pmatrix} b_{11}(t) & b_{12}(t) \\ b_{21}(t) & b_{22}(t) \end{pmatrix}.$$

Now, define:

$$u = \begin{pmatrix} u_{11} \\ u_{21} \\ u_{12} \\ u_{22} \end{pmatrix}, \qquad b(t) = \begin{pmatrix} b_{11}(t) \\ b_{12}(t) \\ b_{21}(t) \\ b_{22}(t) \end{pmatrix},$$

$$\bar{M}_1(t) = \left(\begin{array}{c|c} M(t) & 0 \\ \hline 0 & M(t) \end{array}\right), \qquad \bar{M}_2 = \begin{pmatrix} m_{11}(t) & 0 & m_{21}(t) & 0 \\ 0 & m_{11}(t) & 0 & m_{21}(t) \\ m_{12}(t) & 0 & m_{22}(t) & 0 \\ 0 & m_{12}(t) & 0 & m_{22}(t) \end{pmatrix},$$

and $\bar{M}(t) = \bar{M}_1(t) + \bar{M}_2(t)$. Then it is easy to see that equation (41) is equivalent to:

$$u' = \bar{M}(t)u + b(t). \tag{42}$$

We want to find a T-periodic solution (42) that also satisfies property 2 from Proposition 1. We know that a general solution u of this equation is of the form $u = u_{\mathrm{h}} + u_{\mathrm{p}}$, where u_{p} is a particular solution and u_{h} is the general solution of the homogeneous equation:

$$u' = \bar{M}(t)u.$$

On the other hand, we can write $u_{\mathrm{h}} = \Phi(t)u_0$, where $\Phi(t)$ is the principal fundamental matrix of the homogeneous system (that is a matrix solution such that $\varphi(0) = I_4$, the 4×4 identity matrix) and u_0 is constant. That is:

$$u(t) = \Phi(t)u_0 + u_{\mathrm{p}}(t).$$

For u to be T-periodic, we just need to impose the condition $u(0) = u(T)$. Now, if we take the particular solution u_{p} to be the one such that $u_{\mathrm{p}}(0) = 0$, this condition takes the form:

$$u_0 = \Phi(T)u_0 + u_{\mathrm{p}}(T).$$

That is, the initial condition u_0 must be a solution of:

$$(I_4 - \Phi(T))u_0 = u_{\mathrm{p}}(T). \tag{43}$$

REMARK 16. The matrix $I_4 - \Phi(T)$ is singular (and the reason why property 2 of Proposition 1 is necessary). Indeed, since the matrix $M(t)$ is T-periodic, we know from the Floquet theory that there is a T-periodic solution of the homogeneous system. As a consequence, its monodromy matrix, that is $\Phi(T)$, has an eigenvalue equal to one, that is, $I_4 - \Phi(T)$ has an eigenvalue equal to zero.

However, note that the numerical approximation of $I_4 - \Phi(T)$ will not be a singular matrix. Even so, to avoid the propagation of the error, we will still impose property 2. In the new variables it takes the form:

$$\begin{pmatrix} f^T(x_0(t)) & 0 \\ 0 & f^T(x_0(t)) \end{pmatrix} u(t) + Q^T(t) Df(x_0(t)) = \begin{pmatrix} 0 & 0 \end{pmatrix}.$$

At time $t = 0$ we have:

$$\begin{pmatrix} f^T(x_0(0)) & 0 \\ 0 & f^T(x_0(0)) \end{pmatrix} u(0) + Q^T(0) Df(x_0(0)) = \begin{pmatrix} 0 & 0 \end{pmatrix}. \tag{44}$$

Now we write:

$$A = \begin{pmatrix} I_4 - \Phi(T) \\ \hline f^T(x_0(0)) & 0 \\ 0 & f^T(x_0(0)) \end{pmatrix}, \qquad b = \begin{pmatrix} u_{\mathrm{p}}(T) \\ \hline -Df^T(x_0(0))Q(0) \end{pmatrix}.$$

Note that A is a 4×6 matrix and b a vector from $\mathbb{R}^6$. It is clear from (43) and (44) that the initial condition u_0 must be a solution of the (overdetermined) system:

$$Au_0 = b.$$

Finally, we solve this overdetermined system by least squares, that is the solution u_0 which minimizes

$$||Au_0 - b||.$$

This solution can be found by computing the QR decomposition of the matrix A, obtaining:

$$QRu_0 = b \quad \Rightarrow \quad Ru_0 = Q^T b = \tilde{b},$$

where R is a 4×4 non-singular matrix and $\tilde{b}$ is a vector of $\mathbb{R}^4$ (see [**29**] for more detail). Then we take u_0 as the solution of this compatible system.

Once we have the initial condition u_0 of (41), it lasts only to integrate the system numerically. In this case, for the sake of stability, we have integrated backwards in time.

The implementation of the routine that computes the initial condition for U can be found in Appendix A.9.

1.2. Examples. In this section we will present the results of the computation of $U(t)$ using the method described before. To avoid unnecessary repetitions, we will just show the results (Figures 1, 2, 3 and 4). In each case, we will also show the errors $U(T) - U(0)$ and $Q^T(0)Df^T(x_0(0)) + f^T(x_0(0))U(0)$ to ensure that the least squares solution is reliable.

1.2.1. *Andronov-Hopf oscillator.* Reacall the Andronov-Hopf Oscillator:

$$\begin{cases} \dot{x} &= x - y - x(x^2 + y^2), \\ \dot{y} &= x + y - y(x^2 + y^2). \end{cases} \tag{45}$$

The results obtained for the second order PRCs are (see also Figure 1):

$U(T) - U(0) =$ `[1.93880954e-13, -4.60108396e-10, -4.60107508e-10, -1.92623082e-13]`,

$Q^T(0)Df^T(x_0(0)) + f^T(x_0(0))U(0) =$ `[-6.66133814775094e-16, 1.47924151629920e-15]`.

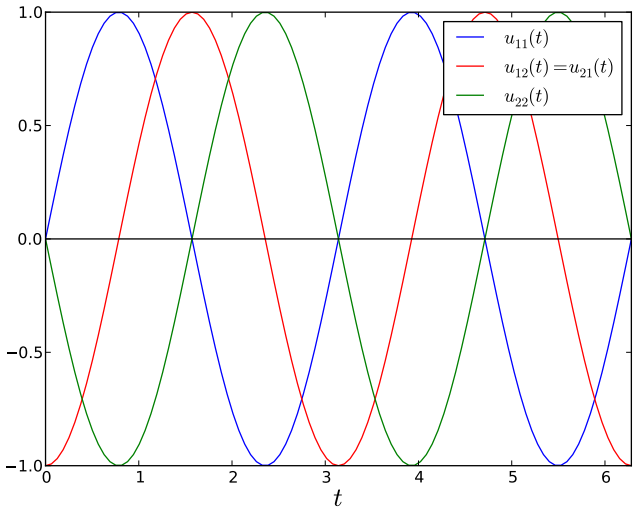


FIG. 1. Solution of the second adjoint equation for the Andronov-Hopf oscillator (45). The first and third curves can be also interpreted as the second order phase advancements when the stimulus directions are $v = (1, 0)$ and $v = (0, 1)$.

1.2.2. *The Selkov model.* The Selkov model was given by the system:

$$\left\{ \begin{array}{rcl} \dot{x} & = & 1 - xy, \\ \dot{y} & = & ay\dfrac{x - (1+b)}{1+by}, \end{array} \right. \tag{46}$$

The results are (see also Figure 2):

$U(T) - U(0) =$ `[2.76161283e-07, 7.19803284e-09, 7.83967291e-09 ,-4.20259383e-10]`,

$Q^T(0)Df^T(x_0(0)) + f^T(x_0(0))U(0) =$ `[4.48565318222904e-9, 2.87049967129249e-9]`.

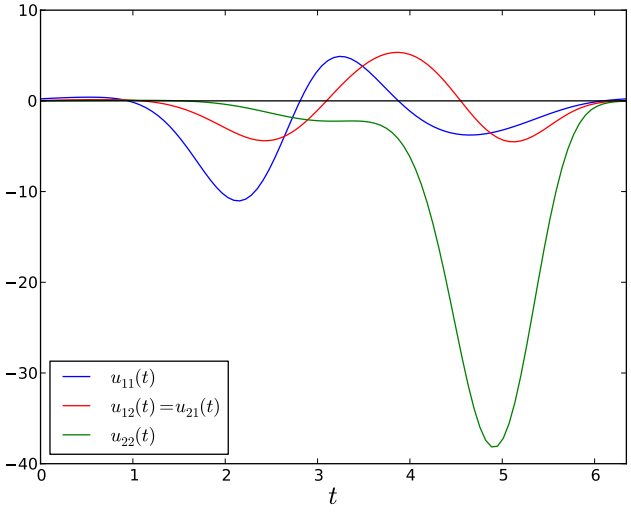


FIG. 2. Solution of the second adjoint equation for the Selkov model (46). The first and third curves can be also interpreted as the second order phase advancements when the stimulus directions are $v = (1, 0)$ and $v = (0, 1)$.

1.2.3. *Reduced Hodgkin-Huxley model.* Recall now the reduced Hodgkin-Huxley model:

$$\left\{\begin{array}{rcl} \dot{V} &=& -\dfrac{1}{C_m}\left[g_{Na}m_\infty(V)(V-V_{Na})\right. \\ & & \left. +g_K n(V-V_k)+g_L(V-V_L)-I_{app}\right], \\ \dot{n} &=& n_\infty(V)-n, \end{array}\right. \tag{47}$$

In this case, we obtained (see also Figure 3):

$U(T)-U(0) =$ `[-5.19367535e-12, 3.26618232e-10, -3.53504864e-09, -1.89463520e-07]`,

$Q^T(0)Df^T(x_0(0))+f^T(x_0(0))U(0) =$ `[ 1.42093403709964e-9, -3.09388521024800e-7]`.

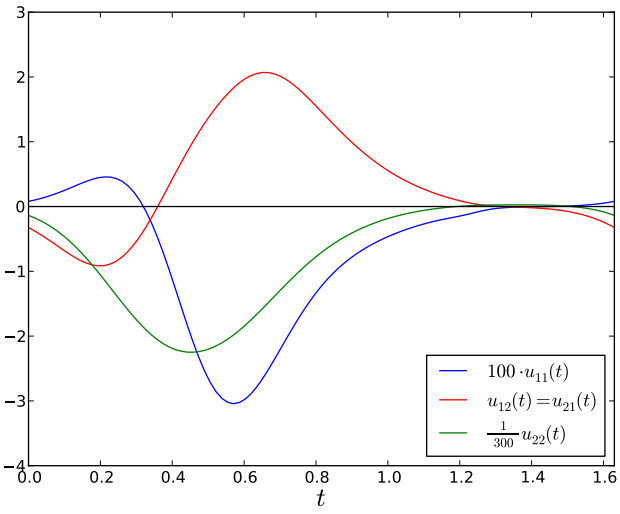


FIG. 3. Solution of the second adjoint equation for the reduced Hodgkin-Huxley model (47) (rescaled). The first and third curves can be also interpreted as the second order phase advancements when the stimulus directions are $v = (1, 0)$ and $v = (0, 1)$.

1.2.4. *The Morris-Lecar model.* Finally we will focus on the Morris-Lecar model, which was given by:

$$(48) \quad \begin{cases} \dot{V} &= \frac{1}{C}\left[I - g_L(V - V_L) - g_K w(V - V_K) - g_{Ca} m_\infty(V)(V - V_{Ca})\right], \\ \dot{w} &= \phi \frac{w_\infty(V) - w}{\tau_w(V)}, \end{cases}$$

The results are (see also Figure 4):

$U(T) - U(0) =$ `[-3.81516325e-11, 5.49064039e-09, 7.03234337e-09, -9.14763575e-07]`,

$Q^T(0) Df^T(x_0(0)) + f^T(x_0(0)) U(0) =$ `[-8.63191809696850e-12, -7.45841024496485e-8]`.

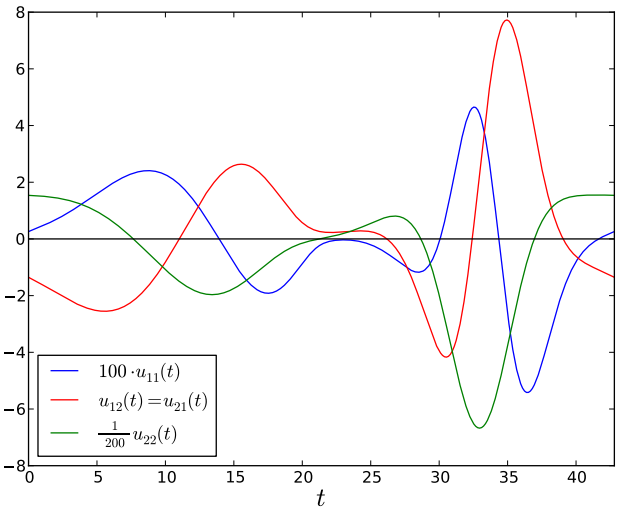


FIG. 4. Solution of the second adjoint equation for the Morris-Lecar model (48) (rescaled). The first and third curves can be also interpreted as the second order phase advancements when the stimulus directions are $v = (1, 0)$ and $v = (0, 1)$.

Note that, in the last two examples, the component u_{22} reaches very large values (compare the scale with the corresponding first order PRCs in Figures 6a and 8a). Thus, it is clear that the consideration of these second order PRCs makes a big difference.

2. Weak perturbations

In this section we will consider again more general perturbations; more precisely, systems of the form:

$$\dot{x} = f(x) + \varepsilon g(x), \tag{49}$$

supposing that for $\varepsilon = 0$ and for ε sufficiently small the system has a periodic solution.

2.1. The Poincaré-Lindstedt method. Let now $x_\varepsilon(t) = \varphi(t, p(\varepsilon), \varepsilon)$ be this $T(\varepsilon)$-periodic solution of (49) with initial condition $p(\varepsilon) \in \Sigma$, where Σ is the isochron of $p(0)$. If we want to approximate this periodic solution up to a given order in the perturbation parameter ε, the first idea that one has is simply to expand this solution:

$$x_\varepsilon(t) = x_0(t) + \varepsilon x_1(t) + \cdots,$$

and try to find the terms of this series. However, if we take into account some terms of order higher than zero, this approximation might not be a uniformly valid

as $\varepsilon \to 0$ over an infinite interval of time. We will illustrate this problem with an example.

EXAMPLE 1. Consider the system:

$$\begin{cases} x' &= x + (1+\varepsilon)y - x(x^2+y^2), \\ y' &= -(1+\varepsilon)x + y - y(x^2+y^2). \end{cases}$$

This system has the periodic orbit:

$$x_\varepsilon(t) = \cos((1+\varepsilon)t), \quad y_\varepsilon(t)) = -\sin((1+\varepsilon)t),$$

which has period $T(\varepsilon) = 2\pi/(1+\varepsilon)$. However, if we expand x and y in their Taylor series with respect to ε we have:

$$x_\varepsilon(t) = \cos t - \varepsilon t \sin t - \frac{1}{2}\varepsilon^2 t^2 \cos t + \cdots,$$

$$y_\varepsilon(t) = -\sin t - \varepsilon t \cos t + \frac{1}{2}\varepsilon^2 t^2 \sin t + \cdots.$$

Hence, the approximations:

$$\hat{x}_\varepsilon(t) = \cos t - \varepsilon t \sin t, \quad \hat{y}_\varepsilon(t) = \sin t - \varepsilon t \cos t,$$

become unbounded as $t \to \infty$, while the true solution remains always bounded. In fact, all approximations obtained by truncating these series at a finite order become unbounded.

Thus, for $\varepsilon \neq 0$, these approximations are not valid for an infinite interval of time, as for $t \to \infty$:

$$|x_\varepsilon(t) - \hat{x}_\varepsilon(t)| \to \infty,$$
$$|y_\varepsilon(t) - \hat{y}_\varepsilon(t)| \to \infty.$$

Another problem that appears when expanding the components of the periodic solution in their Taylor series, and which is also captured in this example, is that their terms are not periodic, except for the first one (although it has period $T(0)$ and not $T(\varepsilon)$ as we would like).

We will now present the Poincaré-Lindstedt method, which allows us to have both a uniformly valid expansion of our periodic solution and with all the terms periodic. We will describe the method through an example, as it is done in [**6**]. For more on the Poincaré-Lindstedt method, we refer the reader also to [**3**].

EXAMPLE 2. Consider the equation:

$$\frac{d^2\bar{x}}{dt^2} + \bar{x} - \varepsilon\bar{x}^3 = 0,$$

with initial conditions $\bar{x}(0) = a$, $d\bar{x}/dt = 0$ at $t = 0$. For $\varepsilon \leq 0$, the solution x_ε with this initial conditions is $T(\varepsilon)-$periodic; for $\varepsilon > 0$ the solution is periodic for $|a| < 1/\sqrt{\varepsilon}$ (see [**6**] for the details).

Let $\omega = \omega(\varepsilon)$ be the unknown frequency of the solution, defined as:

$$\omega = \frac{T(0)}{T(\varepsilon)} = \frac{2\pi}{T(\varepsilon)},$$

so the period is $2\pi/\omega$. Define $\tau = \omega t$ and $x_\varepsilon(\tau) = \bar{x}_\varepsilon(\tau/\omega)$. Then the equation for x_ε reads:

$$\omega^2 \frac{d^2 x_\varepsilon}{d\tau^2} + x_\varepsilon - \varepsilon x_\varepsilon^3 = 0. \tag{50}$$

Note that, for all ε, x_ε is 2π-periodic, since:

$$x_\varepsilon(2\pi) = \bar{x}_\varepsilon(2\pi/\omega) = \bar{x}_\varepsilon(0) = x_\varepsilon(0),$$

and:

$$\frac{dx_\varepsilon}{d\tau}(2\pi) = \frac{1}{\omega}\frac{d\bar{x}_\varepsilon}{dt}(2\pi/\omega) = \frac{1}{\omega}\frac{d\bar{x}_\varepsilon}{dt}(0) = \frac{dx_\varepsilon}{d\tau}(0).$$

Now, we expand the frequency in its Taylor series:

$$\omega = \omega_0 + \varepsilon\omega_1 + \frac{1}{2}\varepsilon^2\omega_2 + \cdots = 1 + \varepsilon\omega_1 + \frac{1}{2}\varepsilon^2\omega_2 + \cdots \tag{51}$$

as well as x_ε:

$$x_\varepsilon(\tau) = x_0(\tau) + \varepsilon x_1(\tau) + \frac{1}{2}\varepsilon^2 x_2(\tau) + \cdots \tag{52}$$

Substituting expressions (51) and (52) into (50), and equating the coefficients of ε^0 we find:

$$\frac{d^2 x_0}{d\tau^2} + x_0 = 0,$$

and $x_0(0) = a$, $dx_0/d\tau = 0$ at $\tau = 0$. Hence:

$$x_0(\tau) = a\cos\tau.$$

Now, equating coefficients of ε and substituting $x_0 = a\cos\tau$, we have that:

$$\frac{d^2 x_1}{d\tau^2} + x_1 = (2\omega_1 a + \frac{3}{4}a^3)\cos\tau + \frac{1}{4}a^3\cos 3\tau,$$

with $x_1(0) = 0$ and $dx_1/d\tau = 0$ at $\tau = 0$. The solution is:

$$x_1(\tau) = \frac{a^3}{32}(\cos\tau - \cos 3\tau) + (2\omega_1 a + \frac{3}{4}a^3)\tau\sin\tau.$$

However, note that the fact that x_ε is 2π-periodic for all ε implies that all of the terms of the expansion (52) also are 2π-periodic. For x_1 to be periodic, ω_1 must be:

$$\omega_1 = -\frac{3}{8}a^2.$$

Then:

$$x_1(\tau) = \frac{a^3}{32}(\cos\tau - \cos 3\tau).$$

This gives the uniformly valid approximation:

$$\bar{x}_\varepsilon(t) = a\cos\omega t + \frac{1}{32}\varepsilon a^3(\cos\tau - \cos 3\tau) + O(\varepsilon^2 a^5)$$
$$\omega = 1 - \frac{3}{8}\varepsilon a^2 + O(\varepsilon^2 a^4) \qquad \text{as } \varepsilon \to 0.$$

REMARK 17. The problem of the Poincaré-Lindstedt method is the difficulty of finding the series explicitly; it can be done for systems of the form:

$$\frac{d^2\bar{x}}{dt^2} + \bar{x} = \varepsilon f\left(\bar{x}, d\bar{x}/dt\right),$$

but for more general systems it is not always possible.

2.2. Second order PRCs for weak perturbations. Let us return to system (49), which had the $T(\varepsilon)$-periodic orbit x_ε. Our purpose now is to obtain a phase model similar to (12) but up to order two. Let ω, τ and $x_\varepsilon(\tau)$ be defined as in the previous section, and define a "re-scaled" phase:

$$\theta(\tau) = \omega\vartheta(\tau/\omega).$$

Note that:

$$\frac{d\theta}{d\tau} = \omega\frac{d\vartheta}{dt}\frac{dt}{d\tau} = \frac{d\vartheta}{dt},$$

and then θ satisfies:

$$\frac{d\theta(x_\varepsilon(\tau))}{d\tau} = 1 + \varepsilon\nabla\vartheta(x_\varepsilon)\cdot g(x_\varepsilon). \tag{53}$$

If we consider the series of x_ε:

$$x_\varepsilon(\tau) = x_0(\tau) + \varepsilon x_1(\tau) + \cdots,$$

and expand the right-hand term of (53) around $x_0(\tau)$, we have:

$$\begin{aligned}\frac{d\theta(x_\varepsilon(\tau))}{d\tau} &= 1 + \varepsilon(\nabla\vartheta(x_0)\cdot g(x_0)) + \\ &+ \frac{1}{2}\varepsilon^2\left[(D\nabla\vartheta(x_0)x_1)\cdot g(x_0) + \nabla\vartheta(x_0)\cdot(Dg(x_0)x_1)\right] + O(\varepsilon^3),\end{aligned} \tag{54}$$

thus obtaining an analogous expression to (12). Note that, as x_0 and x_1 are $T(0)$-periodic, the terms of order ε and ε^2 are $T(0)$-periodic too. For obvious reasons we call the term of order ε^2 the *second order PRC*.

Remark 18. In the following examples we will be able to compute the isochrons analytically, and thus we will know $\nabla\vartheta$ and $D\nabla\vartheta$ explicitly. In case it is not possible, the Adjoint method and the method presented in the previous section can be used to compute these two terms numerically.

Example 3. We will use the example of the nonlinear spring to compute its second order PRC. This system can be written as:

$$\left\{\begin{array}{lcll}\bar{x}' & = & \bar{y}, & \bar{x}(0) = a, \quad \bar{y}(0) = 0.\\ \bar{y}' + \bar{x} - \varepsilon\bar{x}^3 & = & 0, & \end{array}\right. \tag{55}$$

Again, taking $\omega(\varepsilon) = 2\pi/T(\varepsilon)$ the frequency of the perturbed system, we define:

$$x_\varepsilon(\tau) = \bar{x}_\varepsilon(\tau/\omega), \quad y_\varepsilon(\tau) = \bar{y}_\varepsilon(\tau/\omega),$$

and then the first terms of the Poincaré-Lindstedt series are:

$$x_0(\tau) = a\cos\tau, \qquad x_1(\tau) = \frac{a^3}{32}(\cos\tau - \cos 3\tau),$$

$$y_0(\tau) = -a\sin\tau, \qquad y_1(\tau) = \frac{a^3}{32}(3\sin 3\tau - \sin\tau).$$

Recall the phase model (53):

$$\begin{aligned}\frac{d\theta(x_\varepsilon, y_\varepsilon)}{d\tau} &= 1 + \varepsilon(\nabla\vartheta(x_0, y_0)\cdot g(x_0, y_0)) + \\ &+ \frac{1}{2}\varepsilon^2\left[\left(D\nabla\vartheta(x_0, y_0)\begin{pmatrix} x_1 \\ y_1\end{pmatrix}\right)\cdot g(x_0, y_0) + \nabla\vartheta(x_0, y_0)\cdot\left(Dg(x_0, y_0)\begin{pmatrix} x_1 \\ y_1\end{pmatrix}\right)\right] + O(\varepsilon^3).\end{aligned}$$

Let us determine the isochrons of the unperturbed system:

$$\left\{ \begin{array}{rcl} x' & = & y, \\ y' & = & -x, \end{array} \right.$$

Note that this system is not hyperbolic. However, in this case isochrons can be defined. Indeed, in polar coordinates the system takes the form:

$$\left\{ \begin{array}{rcl} r' & = & 0, \\ \vartheta' & = & 1, \end{array} \right. ,$$

so that the isochrons are just straight lines through the origin. Equivalently, the phase ϑ of any point (x, y) is given by:

$$\vartheta(x, y) = \arctan\left(\frac{y}{x}\right),$$

and therefore:

$$\nabla\vartheta(x_0, y_0) = \begin{pmatrix} \dfrac{-y_0}{y_0^2 + x_0^2} \\ \dfrac{x_0}{y_0^2 + x_0^2} \end{pmatrix} = \begin{pmatrix} \dfrac{1}{a}\sin\tau \\ \dfrac{1}{a}\cos\tau \end{pmatrix}.$$

Consequently, we have:

$$D\nabla\vartheta(x_0, y_0) = \begin{pmatrix} \dfrac{2x_0y_0}{(y_0^2 + x_0^2)^2} & \dfrac{-1}{y_0^2 + x_0^2} + \dfrac{2y_0^2}{(y_0^2 + x_0^2)^2} \\ \dfrac{1}{y_0^2 + x_0^2} + \dfrac{-2x_0^2}{(y_0^2 + x_0^2)^2} & \dfrac{-2x_0y_0}{(y_0^2 + x_0^2)^2} \end{pmatrix} =$$

$$= \begin{pmatrix} \dfrac{-1}{a^2}\sin 2\tau & \dfrac{-1}{a^2}\cos 2\tau \\ \dfrac{-1}{a^2}\cos 2\tau & \dfrac{1}{a^2}\sin 2\tau \end{pmatrix}.$$

Finally, noting that:

$$g(x_0, y_0) = \begin{pmatrix} 0 \\ -x_0^3 \end{pmatrix} = \begin{pmatrix} 0 \\ -a^3\cos^3\tau \end{pmatrix},$$

$$Dg(x_0, y_0) = \begin{pmatrix} 0 & 0 \\ -3x_0^2 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ -3a^2\cos^2\tau & 0 \end{pmatrix},$$

we obtain:

$$\begin{aligned} \frac{d\theta}{d\tau} &= 1 - \varepsilon a^2\cos^4\tau + \frac{1}{64}\varepsilon^2 a^4\cos^3\tau\,[\cos 2\tau(\cos\tau - \cos 3\tau) + \\ &\quad + \sin 2\tau(\sin\tau - 3\sin 3\tau) + 3(\cos 3\tau - \cos\tau)] + O(\varepsilon^3). \end{aligned}$$

Note that for large values of a (with respect to ε) the second order PRC becomes significant. In Figure 5 we show an example with $\varepsilon = -0.1$ and $a = 8/\sqrt[3]{|\varepsilon|}$, where it is clear that the second order PRC cannot be ignored.

Example 4. Now we will present an example where the second order PRC plays a crucial role. Take the system:

$$\left\{ \begin{array}{rcl} \dfrac{dx}{dt} & = & (1+\varepsilon)x - y - x(x^2 + y^2), \\ \dfrac{dy}{dt} & = & x + (1+\varepsilon)y - y(x^2 + y^2). \end{array} \right. \tag{56}$$

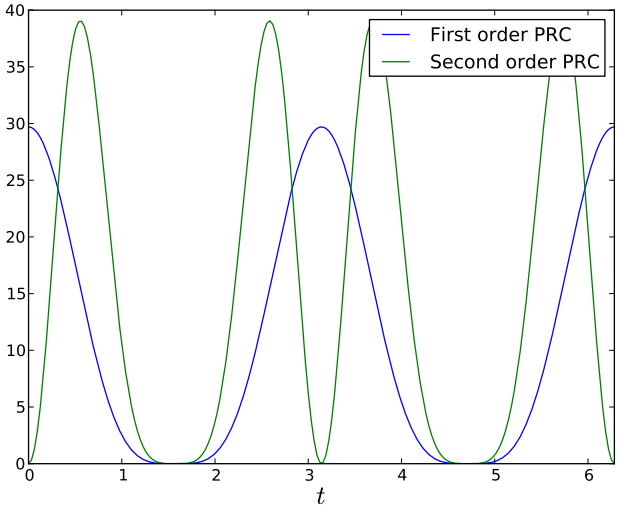


FIG. 5. First and second order PRCs (multiplied by ε and in absolute value) for the nonlinear spring (55), with $\varepsilon = -0.1$ and $a = 8/\sqrt[3]{|\varepsilon|}$.

This system has the periodic orbit:

$$x(t) = \sqrt{1+\varepsilon}\cos t, \qquad y(t) = \sqrt{1+\varepsilon}\sin t.$$

As the period is constant with respect to ε, we have $\omega = 1$ and thus the Poincaré-Lindstedt method is unnecessary in this case. The first terms of the Taylor series of x and y are:

$$x_0(t) = \cos t, \qquad x_1(t) = \frac{1}{2}\cos t,$$
$$y_0(t) = \sin t, \qquad y_1(t) = \frac{1}{2}\sin t.$$

Now, to determine the isochrons of the unperturbed system:

$$\begin{cases} \dfrac{dx}{dt} &= x - y - x(x^2+y^2), \\ \dfrac{dy}{dt} &= x + y - y(x^2+y^2), \end{cases}$$

we will use Remark 4. The field $Y = (x, y)$ and the function $\mu(x, y) = 2(x^2 + y^2)$ satisfy:

$$[Y, f] = \mu Y,$$

where $[\cdot, \cdot]$ is the Lie bracket of two vector fields. We know that the isochrons are the orbits of Y, that is, Y is the tangent vector of the isochrons. As isochrons are the level curves of the function $\vartheta(x, y)$, $\nabla\vartheta(x, y)$ is orthogonal to Y. Finally, using the fact that $\nabla\vartheta(x, y) \cdot f(x, y) = 1$, we conclude that:

$$\vartheta(x, y) = \frac{Y^\perp(x, y)}{Y^\perp(x, y) \cdot f(x, y)}.$$

In this case we have:

$$\nabla\vartheta(x,y) = \begin{pmatrix} \dfrac{-y}{x^2+y^2} \\ \dfrac{x}{x^2+y^2} \end{pmatrix}.$$

Thus:

$$\nabla\vartheta(x_0,y_0) = \begin{pmatrix} -\sin t \\ \cos t \end{pmatrix}.$$

Note that in this case $g(x,y) = (x,y)^T$, and hence the term of order ε of the phase equations (53) is:

$$\nabla\vartheta(x_0,y_0)\cdot g(x_0,y_0) = \begin{pmatrix} -\sin t \\ \cos t \end{pmatrix}\cdot\begin{pmatrix} \cos t \\ \sin t \end{pmatrix} = 0,$$

and hence the term of order ε^2 of (53) governs its dynamics. Let us compute it. We have:

$$D\nabla\vartheta(x,y) = \begin{pmatrix} \dfrac{2xy}{(x^2+y^2)^2} & -\dfrac{1}{(x^2+y^2)} + \dfrac{2y^2}{(x^2+y^2)^2} \\ \dfrac{1}{x^2+y^2} - \dfrac{2x^2}{(x^2+y^2)^2} & \dfrac{-2xy}{(x^2+y^2)^2} \end{pmatrix},$$

and hence:

$$D\nabla\vartheta(x_0,y_0) = \begin{pmatrix} \sin 2t & -\cos 2t \\ -\cos 2t & -\sin 2t \end{pmatrix}.$$

Noting that $Dg(x,y) = I_2$, the 2×2 identity matrix, we have finally that:

$$\frac{d\vartheta}{dt} = 1+\varepsilon^2\left[\frac{1}{2}\cos t\,(\cos t\sin 2t - \sin t\cos 2t) - \frac{1}{2}\sin t\,(\cos t\cos 2t + \sin t\sin 2t)\right] + O(\varepsilon^3).$$

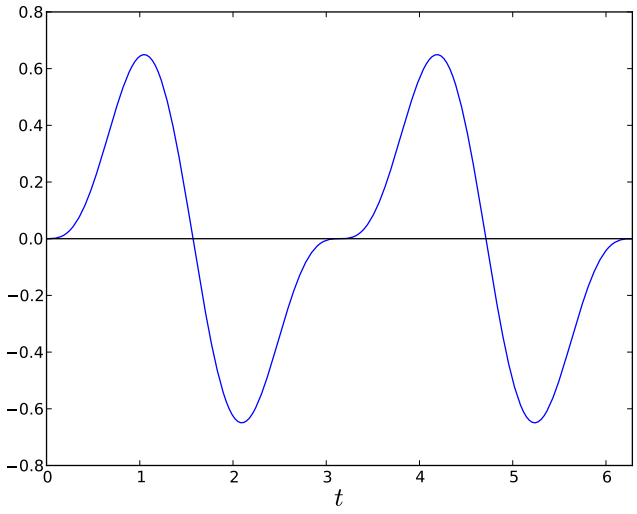


FIG. 6. Second order PRC for system (56).

3. Weak coupling

At this point, we realized that the study of second order PRCs in connected networks of neurons could be interesting, because they can represent the interaction among different neurons through weak synapses. For this reason, now we will consider systems of the form:

$$\dot{x}^i = f_i(x^i) + \varepsilon g_i(x^1, \dots, x^n), \qquad x^i \in \mathbb{R}^2, \quad i = 1, \dots, n, \tag{57}$$

assuming that for $\varepsilon = 0$ each subsystem:

$$x^i = f_i(x^i)$$

has a periodic orbit of an equal fixed period $T_i = T$. We assume also that each subsystem has a $T_i(\varepsilon)$-periodic orbit for ε sufficiently small.

In order to compute the second order PRCs of this system we have to obtain the Poincaré-Lindstedt series of these periodic orbits. However, to be able to do it, the system must have a periodic orbit as a whole. That is, there must be n integers $k_1, \dots, k_n$ such that:

$$k_1 T_1(\varepsilon) = \cdots = k_n T_n(\varepsilon) = T(\varepsilon).$$

To simplify, we will assume that $T_i(\varepsilon) = T(\varepsilon)$ for all i.

Then, the phase model, obtained with a similar procedure as before, is:

(58)

$$\frac{d\theta_i(x^i(\tau))}{d\tau} = 1 + \varepsilon(\nabla\vartheta_i(x_0^i) \cdot g_i(x_0^1, \dots, x_0^n)) + \\ + \frac{1}{2}\varepsilon^2 \left[\left(D\nabla\vartheta_i(x_0^i)x_1^i\right) \cdot g_i(x_0^1, \dots, x_0^n) + \nabla\vartheta_i(x_0^i) \cdot \left(Dg_i(x_0^1, \dots, x_0^n) \begin{pmatrix} x_1^1 \\ \vdots \\ x_1^n \end{pmatrix} \right) \right] + O(\varepsilon^3).$$

Our next example models two coupled oscillators. To obtain the Poincaré-Lindstedt series we used the solution in polar coordinates (and not the method we introduced before); to see how it can be deduced we refer the reader to [**21**].

Example 5. Consider the following system:

$$\begin{cases} \dfrac{d\bar{x}}{dt} &= \bar{x} + \beta\bar{y} - \bar{x}(\bar{x}^2 + \bar{y}^2) + 2\delta(\bar{u} - \bar{x}), \\ \dfrac{d\bar{y}}{dt} &= -\beta\bar{x} + \bar{y} - \bar{y}(\bar{x}^2 + \bar{y}^2) + 2\delta(1 - 2\varepsilon)(\bar{v} - \bar{y}), \\ \dfrac{d\bar{u}}{dt} &= \bar{u} + \beta\bar{u} - \bar{u}(\bar{u}^2 + \bar{v}^2) + 2\delta(\bar{x} - \bar{u}), \\ \dfrac{d\bar{v}}{dt} &= -\beta\bar{u} + \bar{v} - \bar{v}(\bar{u}^2 + \bar{v}^2) + 2\delta(1 - 2\varepsilon)(\bar{y} - \bar{v}). \end{cases} \tag{59}$$

Note that here the perturbation parameter is δ. For $\delta \neq 0$, it can be shown (see [**21**]) that there are two periodic orbits:

$$\gamma_0 = (\bar{x}(t), \bar{y}(t), \bar{u}(t), \bar{v}(t)) = (\cos\beta t, -\sin\beta t, \cos\beta t, -\sin\beta t),$$

and:

$$\gamma_\pi = (\bar{x}(t), \bar{y}(t), \bar{u}(t), \bar{v}(t)) = (\bar{\rho}(t)\cos\bar{\varphi}(t), \bar{\rho}(t)\sin\bar{\varphi}(t), -\bar{\rho}(t)\cos\bar{\varphi}(t), -\bar{\rho}(t)\sin\bar{\varphi}(t)).$$

Note that, in fact, for γ_0 we have $\bar{x} = \bar{u}$, $\bar{y} = \bar{v}$, so that system (59) is the same as it was unperturbed.

Therefore, will only focus on γ_π, which, if $-\beta/4\varepsilon < \delta < \min(1/4(1-\varepsilon), \beta/4\varepsilon)$, is given by:

$$\bar{\rho}(t,\delta) = \sqrt{\frac{[1-4\delta(1-\varepsilon)](1-k^2)}{1+k\cos 2(\bar{\varphi}(t)-\phi)}}, \tag{60}$$

$$k = \frac{4\delta\varepsilon}{\sqrt{[1-4\delta(1-\varepsilon)]^2+\beta^2}}, \qquad \phi = \frac{1}{2}\arctan\frac{-\beta}{1-4\delta(1-\varepsilon)},$$

where $\bar{\varphi}$ is a solution of the equation:

$$\frac{d\bar{\varphi}}{dt} = -\beta + 4\varepsilon\delta\sin 2\bar{\varphi}. \tag{61}$$

The period of this orbit is:

$$T(\delta,\varepsilon,\beta) = \frac{2\pi}{\sqrt{\beta^2-16\delta^2\varepsilon^2}},$$

and then the frequency is:

$$\omega(\delta,\varepsilon,\beta) = \frac{T(0,\varepsilon,\beta)}{T(\delta,\varepsilon,\beta)} = \frac{\sqrt{\beta^2-16\delta^2\varepsilon^2}}{\beta}.$$

Now we do the usual change $x(\tau) = \bar{x}(\tau/\omega)$, $y(\tau) = \bar{y}(\tau/\omega)$, etc., where $\tau = \omega t$, or equivalently:

$$\rho(\tau) = \bar{\rho}(\tau/\omega), \qquad \varphi(\tau) = \bar{\varphi}(\tau/\omega).$$

Note that the terms of the Poincaré-Lindstedt series of x and y are given by:

$$x_0(\tau) = \rho_0(\tau)\cos\varphi_0(\tau), \qquad x_1(\tau) = \rho_1(\tau)\cos\varphi_0(\tau) - \rho_0(\tau)\varphi_1(\tau)\sin\varphi_0(\tau),$$

$$y_0(\tau) = \rho_0(\tau)\sin\varphi_0(\tau), \qquad y_1(\tau) = \rho_1(\tau)\sin\varphi_0(\tau) + \rho_0(\tau)\varphi_1(\tau)\cos\varphi_0(\tau),$$

where ρ_i, φ_i are the corresponding terms of the Poincaré-Lindstedt series of ρ and φ.

Now, taking the derivative of φ with respect to τ and using (61), we have:

$$\omega\frac{d\varphi}{d\tau} = -\beta + 4\varepsilon\delta\sin 2\varphi \quad \Rightarrow \quad (1+\delta\omega_1+\cdots)\left(\frac{d\varphi_0}{d\tau} + \delta\frac{d\varphi_1}{d\tau} + \cdots\right) = -\beta + 4\varepsilon\delta\sin 2\varphi_0 + \cdots.$$

Equating the coefficients of δ^0 and imposing that $\varphi(0) = 0$, we have:

$$\frac{d\varphi_0}{d\tau} = -\beta \quad \Rightarrow \quad \varphi_0(\tau) = -\beta\tau.$$

On the other hand, noting that:

$$\omega_1 = \frac{\partial\omega}{\partial\delta}(0,\varepsilon,\beta) = 0,$$

equating the terms of order δ and keeping in mind that $\varphi(0) = 0$, we obtain:

$$\frac{d\varphi_1}{d\tau} = 4\varepsilon\delta\sin 2\varphi_0 = -4\varepsilon\sin 2\beta\tau \quad \Rightarrow \quad \varphi_1(\tau) = \frac{2\varepsilon}{\beta}\left(\cos 2\beta\tau - 1\right).$$

On the other hand, from (60), we have:

$$\rho_0(\tau) = \rho(\tau, 0) = 1,$$
$$\rho_1(\tau) = \frac{\partial \rho}{\partial \delta}(\tau, 0) = -\frac{1}{2}\left[4(1-\varepsilon) + \frac{4\varepsilon}{\sqrt{1+\beta^2}}\cos\left(2\beta\tau + \arctan(-\beta)\right)\right].$$

Then:

$$x_0(\tau) = \cos\beta\tau,$$
$$x_1(\tau) = -\frac{1}{2}\left[4(1-\varepsilon) + \frac{4\varepsilon}{\sqrt{1+\beta^2}}\cos\left(2\beta\tau + \arctan(-\beta)\right)\right]\cos\beta\tau + \frac{2\varepsilon}{\beta}\left(\cos 2\beta\tau - 1\right)\sin\beta\tau,$$
$$y_0(\tau) = -\sin\beta\tau,$$
$$y_1(\tau) = -\frac{1}{2}\left[4(1-\varepsilon) + \frac{4\varepsilon}{\sqrt{1+\beta^2}}\cos\left(2\beta\tau + \arctan(-\beta)\right)\right]\sin\beta\tau + \frac{2\varepsilon}{\beta}\left(\cos 2\beta\tau - 1\right)\cos\beta\tau.$$

Now we proceed to deduce the isochrons of the first unperturbed system, ϑ_1, in order to determine $\nabla\vartheta_1(x_0, y_0)$. This system is:

$$\left\{\begin{array}{rcl} \dfrac{d\bar{x}}{dt} &=& \bar{x} + \beta\bar{y} - \bar{x}(\bar{x}^2 + \bar{y}^2), \\ \dfrac{d\bar{y}}{dt} &=& -\beta\bar{x} + \bar{y} - \bar{y}(\bar{x}^2 + \bar{y}^2). \end{array}\right. \tag{62}$$

Recalling again Remark 4, we can use it to compute the isochrons. Indeed, it is easy to see that the field $Y(x, y) = (x, y)$ and $\mu(x) = -2(x^2 + y^2)$ satisfy the property:

$$[Y, f_1] = \mu Y.$$

Hence:

$$\nabla\vartheta_1(x, y) = \frac{Y^\perp}{Y^\perp \cdot f_1} = \begin{pmatrix} \dfrac{y}{\beta(x^2+y^2)} \\ \dfrac{-x}{\beta(x^2+y^2)} \end{pmatrix}.$$

$$D\nabla\vartheta_1(x, y) = \begin{pmatrix} \dfrac{-2xy}{\beta(y^2+x^2)^2} & \dfrac{1}{\beta(y^2+x^2)} - \dfrac{2y^2}{\beta(y^2+x^2)^2} \\ \dfrac{-1}{\beta(y^2+x^2)} + \dfrac{2x^2}{\beta(y^2+x^2)^2} & \dfrac{2xy}{\beta(y^2+x^2)^2} \end{pmatrix}.$$

Then, we have that:

$$\nabla\vartheta_1(x_0, y_0) = \begin{pmatrix} -\dfrac{1}{\beta}\sin\beta\tau \\ -\dfrac{1}{\beta}\cos\beta\tau \end{pmatrix},$$

$$D\nabla\vartheta_1(x_0, y_0) = \begin{pmatrix} \dfrac{1}{\beta}\sin 2\beta\tau & \dfrac{1}{\beta}\cos 2\beta\tau \\ \dfrac{1}{\beta}\cos 2\beta\tau & -\dfrac{1}{\beta}\sin 2\beta\tau \end{pmatrix}.$$

On the other hand, recall that the perturbation of the first subsystem is:

$$g_1(x,y,u,v) = \begin{pmatrix} 2(u-x) \\ 2(1-2\varepsilon)(v-y) \end{pmatrix}.$$

Then, since $u_0(\tau) = -x_0(\tau)$ and $v_0(\tau) = -y_0(\tau)$, we find:

$$g_1(x_0,y_0,u_0,v_0) = \begin{pmatrix} -4\cos\beta\tau \\ 4(1-2\varepsilon)\sin\beta\tau \end{pmatrix},$$

$$Dg_1(x_0,y_0,u_0,v_0) = \begin{pmatrix} -4 & 0 & 4 & 0 \\ 0 & -4(1-2\varepsilon) & 0 & 4(1-2\varepsilon) \end{pmatrix}.$$

Finally, from (58), we obtain the phase model for the first subsystem:

$$\begin{aligned} \frac{d\theta_1}{d\tau} &= 1 + \frac{1}{\beta}\delta\left[4\sin\beta\tau\cos\beta\tau - 4(1-2\varepsilon)\sin\beta\tau\cos\beta\tau\right] + \frac{1}{2\beta}\delta^2\left[-4\cos\beta\tau\,(x_1\sin 2\beta\tau + \right. \\ &\quad + y_1\cos 2\beta\tau) + 4(1-2\varepsilon)\sin\beta\tau\,(x_1\cos 2\beta\tau - y_1\sin 2\beta\tau) + 8x_1\sin\beta\tau \\ &\quad \left. + 8(1-2\varepsilon)y_1\cos\beta\tau\right] + O(\delta^3). \end{aligned}$$

The phase model of the second subsystem can be obtained in an analogous way.

Conclusions

In this memoir we have tried to give an overview of some applications of dynamical systems in the context of neuroscience. First, we explained the concepts of asymptotic phase and isochron, which yield to the introduction of PRCs. We have also explained how they are used in practice, as well as their applications in synchrony problems. During the presentation of this theoretical background we have found many mathematical techniques, which go from Lie symmetries to averaging theory.

Next, we have described the numerical methods to compute the objects introduced in the first chapter, also programming them in Sage. Moreover, we have contributed to the development of this mathematical software by creating a routine that will be in its next release, and creating a package of an already existing program, Brian. Thus, we do not just contribute to the mathematical community but also to the neuroscientific, providing it with a new and very powerful platform in which to run Brian.

Finally, we have introduced the second order PRCs. In the case of weak coupling, we have derived a new method to compute them numerically. In the case of weak perturbations, we have introduced a different framework from the classical one that requires Poincaré-Lindsted series. In all cases, we have tried to give examples to illustrate the importance of considering these second order terms.

As future work, it would be very interesting to do second order averaging in the phase equations, which is possible in this new framework, because the second order PRCs are periodic. Besides, it would be nice to translate second order PRCs in the language of Lie symmetries, because it would provide them with a geometrical meaning and could derive new methods for their computation. It could be also useful to study more deeply examples in which second order PRCs become significant, giving a biological explanation of it in the case of neuronal models, and also consider examples where the perturbation depends on ε in a general way (in this case, second order PRCs would need to be deduced again in a similar way we have done). Finally, second order PRCs of pulsed coupled systems could be used to find second approximations of isochrons, in a similar way as in [**30**]. In conclusion, we have opened a new path in which there is still a lot to discover.

Appendix A
Code

In this appendix we enclose the main routines we have created and which have been exposed in this memoir. The code of some subroutines, which are not particularly relevant, has not been added to avoid a number of unnecessary pages. For the same reason, the documentation of the routines has been suppressed, as we think its use has been clarified enough through the examples presented in the previous sections.

1. Poincaré map

```
def poincare(ff,T,Tneg,ci):
    section=ci[1]

    # Integrate one step
    T.y_0=ci
    step=float(0.01)
    T.ode_solve(t_span=[0,step],num_points=2)
    yd=[points for [tm,points] in T.solution]
    yd=numpy.array(yd)
    time=step
    sol=yd[-1][:]
    g=sol[1]-section
    g_old=g
    dg=[0,1]
    sign=sgn(g)

# Integrate until the solution has crossed the Poincare section twice
    while g*g_old>0 or g*sign<0 :
        T.y_0=sol
        T.ode_solve(t_span=[0,step],num_points=2)
        yd=[points for [tm,points] in T.solution]
        yd=numpy.array(yd)
        time+=step
        sol=yd[-1][:]
        g_old=g
```

```
            g=sol[1]-section

        # Refine with Newton's method
        while abs(g)>1e-13:
            camp=[ff[0](*sol),ff[1](*sol)][:]
            step=-(sol[1]-section)/camp[1]
            if step>=0:
                T.y_0=sol
                T.ode_solve(t_span=[0,step],num_points=100)
                yd=[points for [tm,points] in T.solution]
                yd=numpy.array(yd)
            else:
                Tneg.y_0=sol
                Tneg.ode_solve(t_span=[0,-step],num_points=100)
                yd=[points for [tm,points] in Tneg.solution]
                yd=numpy.array(yd)
            time+=step
            sol=yd[-1][:]
            g=sol[1]-section
        return time, sol
```

2. Differential of the Poincaré map

```
def dif_poincare(ff,T1,T1neg,T2,x):
    temps,p0=poincare(ff,T1,T1neg,x)
    dg=matrix([0,1])
    dphi,p1=variational(T2,x,temps)
    fp=matrix([[ff[0](*p0)],[ff[1](*p0)]])
    num=dg*dphi
    div=dg*fp
    div=div[0,0]
    dp=-fp*num/div+dphi
    return dp[0,0]
```

3. Periodic orbit finder

```
def find_po(f,ci):
    ff=fast_float(f,'x','y')
    gg=create_func_variational_poincare(f,(x,y))
    field=f[:]
    fneg=[-fu for fu in f]
    field.extend(gg)
    T1=create_odesolver(f,[x,y],algorithm="rk8pd",error_abs=1e-17,
    error_rel=1e-17)
    T1neg=create_odesolver(fneg,[x,y],algorithm="rk8pd",error_abs=1e-17,
    error_rel=1e-17)
```

```
    T2=create_odesolver(field,[x,y,s11,s12,s21,s22],algorithm="rk8pd",
    error_abs=1e-17,error_rel=1e-17)
    period,s=poincare(ff,T1,T1neg,ci)
    while (abs(s[0]-ci[0])>1e-16):
        step=-(s[0]-ci[0])/(dif_poincare(ff,T1,T1neg,T2,ci)-1)
        ci[0]=ci[0]+step
        period,s=poincare(ff,T1,T1neg,ci)
    return ci,period
```

4. Adjoint system solver

```
def adjoint(f,ci,period,numpoints=100,vars=[x,y]):
    if vars!=[x,y] and vars!=[y,x]:
        f[0]=f[0].subs_expr(vars[0]==x,vars[1]==y)
        f[1]=f[1].subs_expr(vars[0]==x,vars[1]==y)

    ff=fast_float(f,'x','y')

    # Create the matricial and vectorial adjoint function
    g=create_func_adjoint_mat(f)
    h=create_func_adjoint_vec(f)

    # Set initial conditions of the adjoint system
    ci=list((ci[0],ci[1],1,0,0,1))

    # Define the fields
    g.insert(0,f[0])
    g.insert(1,f[1])
    h.insert(0,f[0])
    h.insert(1,f[1])

    # Integrate
    T=create_odesolver(g,[x,y,s11,s12,s21,s22],y_0=ci,algorithm="rk8pd",
    error_abs=1e-17,error_rel=1e-17)
    T.ode_solve(t_span=[0,period],num_points=numpoints)
    xd=[time for [time,points] in T.solution]
    yd=[points for [time,points] in T.solution]
    yd=numpy.array(yd)

    M=numpy.matrix([[yd[-1][2],yd[-1][3]],[yd[-1][4],yd[-1][5]]])

    vep=numpy.array([])
    vep.resize(2)
    vaps=numpy.linalg.eig(M)[0]
    if (abs(vaps[0]-1)<abs(vaps[1]-1)):
        vep[0]=numpy.linalg.eig(M)[1][0,0]
        vep[1]=numpy.linalg.eig(M)[1][1,0]
```

```
    else:
        vep[0]=numpy.linalg.eig(M)[1][0,1]
        vep[1]=numpy.linalg.eig(M)[1][1,1]

    norm=ff[0](*ci[0:2])*vep[0]+ff[1](*ci[0:2])*vep[1]
    ci=(ci[0],ci[1],vep[0]/norm,vep[1]/norm)

    T=create_odesolver(h,[x,y,u,v],y_0=ci,algorithm="rk8pd",
    error_abs=1e-16,error_rel=1e-16)
    T.ode_solve(t_span=[0,period],num_points=numpoints)
    yd=[points for [time,points] in T.solution]
    yd=numpy.array(yd)

    return [xd,yd]
```

5. Interaction function

```
def interaction(f,g12,ci,period,npoints=100):
    # Compile g12
    gg120=fast_float(g12[0],'x1','y1','x2','y2')
    gg121=fast_float(g12[1],'x1','y1','x2','y2')

    # Compute the PRC
    [xd,yd]=adjoint(f,ci,period,numpoints=npoints)
    Q=yd[:,2:4]

    sol=yd[:,0:2]
    H=numpy.array([])
    H.resize(npoints+1)
    solshift=numpy.array([])
    solshift.resize(npoints,2)

    for phshift in range(0,npoints):
        # Compute the solution with phase shift phshift
        ci=sol[phshift]
        solshift=desolve_odeint(f,ci,xd,[x,y],rtol=1e-13,atol=1e-14)

        # Compute the function to be integrated
        gval0=map(gg120,sol[:,0],sol[:,1],solshift[:,0],solshift[:,1])
        gval1=map(gg121,sol[:,0],sol[:,1],solshift[:,0],solshift[:,1])
        val=Q[:,0]*gval0+Q[:,1]*gval1

        # Integrate using the composite trapezoidal rule
        H[phshift]=1/period*integrate.simps(val,dx=xd[1])

    return xd,H
```

6. Synchronization function

```
def synch(f,g12,ci,period,npoints=100):
    xd,H=interaction(f,g12,ci,period,npoints=npoints)
    G=numpy.array([])
    G.resize(npoints+1)

    for i in range(0,npoints):
        G[i]=H[-(i+1)]-H[i]

    return xd, H, G
```

7. desolve_odeint

```
def desolve_odeint(des, ics, times, dvars, ivar=None, compute_jac=False,
args=(), rtol=None, atol=None, tcrit=None, h0=0.0, hmax=0.0, hmin=0.0,
ixpr=0, mxstep=0, mxhnil=0, mxordn=12, mxords=5, printmessg=0):

    from scipy.integrate import odeint
    from sage.ext.fast_eval import fast_float
    from sage.calculus.functions import jacobian

    if ivar==None:
        if len(dvars)==0 or len(dvars)==1:
            if len(dvars)==1:
                des=des[0]
                dvars=dvars[0]

            all_vars = set(des.variables())
        else:
            all_vars = set([])
            for de in des:
                all_vars.update(set(de.variables()))
        if is_SymbolicVariable(dvars):
            ivars = all_vars - set([dvars])
        else:
            ivars = all_vars - set(dvars)

        if len(ivars)==1:
            ivar = ivars.pop()
        elif not ivars:
            from sage.symbolic.ring import var
            safe_name = 't_' + str(dvars)
            ivar = var(safe_name)
        else:
            raise ValueError
            "Unable to determine independent variable, please specify."
```

```
        # one-dimensional systems:
        if is_SymbolicVariable(dvars):
            func = fast_float(des,dvars,ivar)
            if not compute_jac:
                Dfun=None
            else:
                J = diff(des,dvars)
                J = fast_float(J,dvars,ivar)
                Dfun = lambda y,t: [J(y,t)]

        # n-dimensional systems:
        else:
            desc = []
            variabs = dvars[:]
            variabs.append(ivar)
            for de in des:
                desc.append(fast_float(de,*variabs))

            def func(y,t):
                v = list(y[:])
                v.append(t)
                return [dec(*v) for dec in desc]

            if not compute_jac:
                Dfun=None
            else:
                J = jacobian(des,dvars)
                J = [list(v) for v in J]
                J = fast_float(J,*variabs)
                def Dfun(y,t):
                    v = list(y[:])
                    v.append(t)
                    return [[element(*v) for element in row] for row in J]

        sol=odeint(func, ics, times, args=args, Dfun=Dfun, rtol=rtol,
        atol=atol, tcrit=tcrit, h0=h0, hmax=hmax, hmin=hmin, ixpr=ixpr,
        mxstep=mxstep, mxhnil=mxhnil, mxordn=mxordn, mxords=mxords,
        printmessg=printmessg)

        return sol
```

8. create_odesolver

```
def create_odesolver(f,dvars,ivar=None,a=None,a_dydt=None,algorithm=None,
error_abs=None,error_rel=None,h=None,params=None,scale_abs=None,y_0=None):
```

```
if algorithm=="bsimp":
    compute_jac=True
else:
    compute_jac=False

if ivar==None:
    from sage.symbolic.ring import var
    ivar=var('t')

# one-dimensional systems:
if len(dvars)==1 or len(dvars)==0:
    if type(f)==list or type(f)==tuple:
        f=f[0]

    fc=fast_float(f,ivar,dvars)
    func=lambda t,y: [fc(t,y[0])]
    if type(y_0)!=list and type(y_0)!=tuple:
        y_0=[y_0]

    if compute_jac==False:
        Dfun=None
    else:
        J=jacobian(f,dvars)
        J=J[0][0]
        J=[[fast_float(J,ivar,dvars)],[0]]
        Dfun=lambda t,y: J(t,y)


# n-dimensional systems:
else:
    fc=[]
    variabs=[v for v in dvars]
    variabs.insert(0,ivar)
    for de in f:
        fc.append(fast_float(de,*variabs))

    def func(t,y):
        v=[p for p in y]
        v.insert(0,t)
        val=[dec(*v) for dec in fc]
        return val

    if compute_jac==False:
        Dfun=None
    else:
        J=jacobian(f,dvars)
        J=[list(v) for v in J]
        J=fast_float(J,*variabs)
```

```
            def Dfun(t,y):
                v=[p for p in y]
                v.insert(0,t)
                D=[]
                z=[]
                for row in J:
                    r=[]
                    z.append(0)
                    for element in row:
                        r.append(element(*v))

                    D.append(r)
                D.append(z)

                return D

    T=ode_solver()
    T.function=func
    if a!=None:
        T.a=a
    if a_dydt!=None:
        T.a_dydt=a_dydt
    if algorithm!=None:
        T.algorithm=algorithm
    if error_abs!=None:
        T.error_abs=error_abs
    if error_rel!=None:
        T.error_rel=error_rel
    if h!=None:
        T.h=h
    if params!=None:
        T.params=params
    if compute_jac==True:
        T.jacobian=Dfun
    if scale_abs!=None:
        T.scale_abs=scale_abs
    if y_0!=None:
        T.y_0=y_0

    return T
```

9. Computation of the initial conditions of $D\nabla\vartheta$

```
def find_ic_dq(f,ci,period,numpoints):
    df=jacobian(f,(x,y))
    A=-transpose(df)
```

```
m1aux1=A.augment(zero_matrix(SR,2,2))
m1aux2=zero_matrix(SR,2,2).augment(A)
M1=m1aux1.stack(m1aux2)

A=transpose(A)
M2=matrix(SR,[[A[0,0], 0, A[1,0], 0],[0, A[0,0], 0, A[1,0]],[A[0,1],
0, A[1,1], 0],[0, A[0,1], 0, A[1,1]]])

M=M1+M2

# Find the monodromy matrix
var('v11,v12,v13,v14,v21,v22,v23,v24,v31,v32,v33,v34,v41,v42,v43,v44')
X=matrix(SR,[[v11,v12,v13,v14],[v21,v22,v23,v24],[v31,v32,v33,v34],
[v41,v42,v43,v44]])
ffun=M*X
ffun=[ffun[0,0],ffun[0,1],ffun[0,2],ffun[0,3],ffun[1,0],ffun[1,1],
ffun[1,2],ffun[1,3],ffun[2,0],ffun[2,1],ffun[2,2],ffun[2,3],ffun[3,0],
ffun[3,1],ffun[3,2],ffun[3,3]]

field=f[:]
h=create_func_adjoint_vec(f)
field.extend(h)
field.extend(ffun)
cifield=ci[:]
cifield.extend([1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1])
from scipy import linspace
xd=linspace(0,period,numpoints)

T=create_odesolver(field,[x,y,u,v,v11,v12,v13,v14,v21,v22,v23,v24,v31,
v32,v33,v34,v41,v42,v43,v44],y_0=cifield,algorithm="rk8pd",
error_abs=1e-17,error_rel=1e-17)
T.ode_solve(t_span=[0,period],num_points=numpoints)
fundmatrix=[points for [time,points] in T.solution]
fundmatrix=numpy.array(fundmatrix)

monodromy=fundmatrix[-1,4:20]
monodromy=matrix(RDF,4,list(monodromy))
A2=identity_matrix(RDF,4)-monodromy

vec1=matrix([[f[0].subs(x=ci[0],y=ci[1]),f[1].subs(x=ci[0],
y=ci[1])]]).augment(zero_matrix(1,2))
vec2=zero_matrix(RDF,1,2).augment(matrix([[f[0].subs(x=ci[0],y=ci[1])
,f[1].subs(x=ci[0],y=ci[1])]]))

A2=A2.stack(vec1)
A2=A2.stack(vec2)

# Particular solution
```

```
s=create_func_s(f)
ciparticular=ci[:]
ciparticular.extend([0,0,0,0])
fpart=field[0:4]
fpart.extend([s[0],s[2],s[1],s[3]])

T=create_odesolver(fpart,[x,y,u,v,s11,s21,s12,s22],y_0=ciparticular,
algorithm="rk8pd",error_abs=1e-17,error_rel=1e-17)
T.ode_solve(t_span=[0,period],num_points=numpoints)
sol=[points for [time,points] in T.solution]
sol=numpy.array(sol)

solpart=sol[-1,4:8]
indepterm=list(solpart[:])
aux=matrix(RDF,[[-ci[2],-ci[3]]])
aux=aux*df(x=ci[0],y=ci[1])
indepterm.extend(list(aux[0]))
indepterm=vector(RDF,indepterm)

Q,R=A2.QR()
var('c11,c21,c12,c22')
mat=R[0:4,:]
vec=transpose(Q)*indepterm
syseq=mat*vector([c11,c21,c12,c22])
ics=solve([syseq[0]==vec[0],syseq[1]==vec[1],syseq[2]==vec[2],
syseq[3]==vec[3]],c11,c21,c12,c22,solution_dict=true)
ics=[RDF(ics[0][c11]), RDF(ics[0][c12]), RDF(ics[0][c21]), RDF(ics[0][c22])]

return ics
```

References

1. E. Brown, J. Moehlis, and P. Holmes, *On the phase reduction and response dynamics of neural oscillator populations*, Neural Computation **16** (2004), no. 4, 673–715.
2. C. Cajochen, K. Kräuchi, and A. Wirz-Justice, *Role of melatonin in the regulation of human circadian rhythms and sleep*, J. Neuroendocrinology **15** (2003), 432–437.
3. C. Chicone, *Ordinary differential equations with applications*, Springer Verlag, 2006.
4. C. Chicone and W. Liu, *Asymptotic phase revisited*, Journal of Differential Equations **204** (2004), no. 1, 227–246. MR MR2076165 (2005e:34085)
5. A. Diez-Noguera, *A functional model of the circadian system based on the degree of intercommunication in a complex system*, Am J Physiol Regul Integr Comp Physiol **267** (1994), R1118–R1135.
6. PG Drazin, *Nonlinear systems*, Cambridge University Press, 1992.
7. F. Dumortier, *Asymptotic phase and invariant foliations near periodic orbits*, Proceedings of the American Mathematical Society **134** (2006), no. 10, 2989–2996.
8. G.B. Ermentrout and D.H. Terman, *Mathematical Foundations of Neuroscience*, Springer, New York, 2010.
9. M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, *GNU Scientific Library Reference Manual, Revised Second Edition (v1. 8)*, Network Theory Ltd, 2006.
10. D.F. Goodman and R. Brette, *Brian: a simulator for spiking neural networks in Python*, Frontiers in Neuroinformatics **2** (2008), no. 5.
11. ———, *The Brian simulator*, Frontiers in Neuroscience **3** (2009), no. 2, 192–197.
12. J. Guckenheimer, *Isochrons and phaseless sets*, Journal of Mathematical Biology **1** (1975), no. 3, 259–273.
13. A. Guillamon and G. Huguet, *A computational and geometric approach to phase resetting curves and surfaces*, SIAM Journal on Applied Dynamical Systems **8** (2009), no. 3, 1005–1042. MR 2551254
14. E. Hairer, S.P. Nørsett, and G. Wanner, *Solving ordinary differential equations I: Nonstiff problems*, Springer, 1993.
15. A.C. Hindmarsh, *ODEPACK, a systematized collection of ODE solvers*, Scientific Computing **1** (1983), 55–64.
16. F.C. Hoppensteadt and E.M. Izhikevich, *Weakly connected neural networks*, Springer Verlag, 1997.
17. E.M. Izhikevich, *Dynamical systems in neuroscience*, MIT Press, 2007.
18. Y. Kuramoto, *Cooperative dynamics of oscillator community*, Prog. Theoret. Phys. Suppl **79** (1984), 223–240.
19. C. Morris and H. Lecar, *Voltage oscillations in the barnacle giant muscle fiber*, Biophys. J. **35** (1981), 193–213.
20. B. O'Sullivan, *Mercurial: The definitive guide*, Oreilly & Associates Inc, 2009.
21. H.G. Othmer and M. Watanabe, *On the collapse of the resonance structure in a three-parameter family of coupled oscillators*, Rocky Mountain J. Math **18** (1988), no. 2, 403–432.
22. A.A. Ptitsyn, S. Zvonic, and J.M. Gimble, *Digital signal processing reveals circadian baseline oscillation in majority of mammalian genes*, PLoS Comput Biol **3** (2007), no. 6.
23. E.L. Reiss, *A new asymptotic method for jump phenomena*, SIAM Journal on Applied Mathematics **39** (1980), no. 3, 440–455.

24. J. Rinzel and G.B. Ermentrout, *Analysis of neural excitability and oscillations*, Methods in Neural Modeling (Cambridge, MA) (C. Koch and I. Segev, eds.), MIT Press., 1998, pp. 135–169.
25. M. Sabatini, *Isochronous sections via normalizers*, International Journal of Bifurcation and Chaos in Applied Sciences and Engineering **15** (2005), no. 9, 3031–3037.
26. J.A. Sanders and F. Verhulst, *Averaging methods in nonlinear dynamical systems*, Applied Mathematical Sciences, volume 59, Springer-Verlag, 1985.
27. E. Selkov, *On the mechanism of single-frequency self-oscillations in glycolysis. A simple kinetic model*, Eur J Biochem **4** (1968), 79–86.
28. W. A. Stein et al., *Sage Mathematics Software (Version 4.5)*, The Sage Development Team, 2010, `http://www.sagemath.org`.
29. J. Stoer and R. Bulirsch, *Introduction to numerical analysis*, Springer Verlag, 2002.
30. D. Takeshita and R. Feres, *Higher order approximation of isochrons*, Nonlinearity **23** (2010), 1303–1323.